AMD 64-Bit Technology

AMD x86-64 Architecture Programmer's Manual

Volume 5:

64-Bit Media and x87 Floating-Point Instructions

Publication No.	Revision	Date
26569	**3.00**	**August 2002**

Contents

Figures

Tables

Preface

About This Book

This book is part of a multivolume work entitled the *AMD x86-64 Architecture Programmer's Manual*. This table lists each volume and its order number.

Title	Order No.
Volume 1, *Application Programming*	24592
Volume 2, *System Programming*	24593
Volume 3, *General-Purpose and System Instructions*	24594
Volume 4, *128-Bit Media Instructions*	26568
Volume 5, *64-Bit Media and x87 Floating-Point Instructions*	26569

Audience

This volume (Volume 5) is intended for all programmers writing application or system software for a processor that implements the x86-64 architecture.

Organization

Volumes 3, 4, and 5 describe the x86-64 architecture's instruction set in detail. Together, they cover each instruction's mnemonic syntax, opcodes, functions, affected flags, and possible exceptions.

The x86-64 instruction set is divided into five subsets:

- General-purpose instructions
- System instructions
- 128-bit media instructions
- 64-bit media instructions
- x87 floating-point instructions

Several instructions belong to—and are described identically in—multiple instruction subsets.

This volume describes the 64-bit media and x87 floating-point instructions. The index at the end cross-references topics within this volume. For other topics relating to the x86-64 architecture, and for information on instructions in other subsets, see the tables of contents and indexes of the other volumes.

Definitions

Many of the following definitions assume an in-depth knowledge of the legacy x86 architecture. See “Related Documents” on page xxiii for descriptions of the legacy x86 architecture.

Terms and Notation

In addition to the notation described below, “Opcode-Syntax Notation” in volume 3 describes notation relating specifically to opcodes.

1011b

A binary value—in this example, a 4-bit value.

F0EAh

A hexadecimal value—in this example a 2-byte value.

[1,2)

A range that includes the left-most value (in this case, 1) but excludes the right-most value (in this case, 2).

7–4

A bit range, from bit 7 to 4, inclusive. The high-order bit is shown first.

128-bit media instructions

Instructions that use the 128-bit XMM registers. These are a combination of the SSE and SSE2 instruction sets.

64-bit media instructions

Instructions that use the 64-bit MMX™ registers. These are primarily a combination of MMX and 3DNow!™ instruction sets, with some additional instructions from the SSE and SSE2 instruction sets.

16-bit mode

Legacy mode or compatibility mode in which a 16-bit address size is active. See *legacy mode* and *compatibility mode*.

32-bit mode

Legacy mode or compatibility mode in which a 32-bit address size is active. See *legacy mode* and *compatibility mode*.

64-bit mode

A submode of *long mode*. In 64-bit mode, the default address size is 64 bits and new features, such as register extensions, are supported for system and application software.

#GP(0)

Notation indicating a general-protection exception (#GP) with error code of 0.

absolute

Said of a displacement that references the base of a code segment rather than an instruction pointer. Contrast with *relative*.

biased exponent

The sum of a floating-point value's exponent and a constant bias for a particular floating-point data type. The bias makes the range of the biased exponent always positive, which allows reciprocation without overflow.

byte

Eight bits.

clear

To write a bit value of 0. Compare *set*.

compatibility mode

A submode of *long mode*. In compatibility mode, the default address size is 32 bits, and legacy 16-bit and 32-bit applications run without modification.

commit

To irreversibly write, in program order, an instruction's result to software-visible storage, such as a register (including flags), the data cache, an internal write buffer, or memory.

CPL

Current privilege level.

CR0–CR4

A register range, from register CR0 through CR4, inclusive, with the low-order register first.

CR0.PE = 1

Notation indicating that the PE bit of the CR0 register has a value of 1.

direct

Referencing a memory location whose address is included in the instruction's syntax as an immediate operand. The address may be an absolute or relative address. Compare *indirect*.

dirty data

Data held in the processor's caches or internal buffers that is more recent than the copy held in main memory.

displacement

A signed value that is added to the base of a segment (absolute addressing) or an instruction pointer (relative addressing). Same as *offset*.

doubleword

Two words, or four bytes, or 32 bits.

double quadword

Eight words, or 16 bytes, or 128 bits. Also called *octword*.

DS:rSI

The contents of a memory location whose segment address is in the DS register and whose offset relative to that segment is in the rSI register.

EFER.LME = 0

Notation indicating that the LME bit of the EFER register has a value of 0.

effective address size

The address size for the current instruction after accounting for the default address size and any address-size override prefix.

effective operand size

The operand size for the current instruction after accounting for the default operand size and any operand-size override prefix.

element

See *vector.*

exception

An abnormal condition that occurs as the result of executing an instruction. The processor's response to an exception depends on the type of the exception. For all exceptions except 128-bit media SIMD floating-point exceptions and x87 floating-point exceptions, control is transferred to the handler (or service routine) for that exception, as defined by the exception's vector. For floating-point exceptions defined by the IEEE 754 standard, there are both masked and unmasked responses. When unmasked, the exception handler is called, and when masked, a default response is provided instead of calling the handler.

FF /0

Notation indicating that FF is the first byte of an opcode, and a subfield in the second byte has a value of 0.

flush

An often ambiguous term meaning (1) writeback, if modified, and invalidate, as in "flush the cache line," or (2) invalidate, as in "flush the pipeline," or (3) change a value, as in "flush to zero."

GDT

Global descriptor table.

IDT

Interrupt descriptor table.

IGN

Ignore. Field is ignored.

indirect

Referencing a memory location whose address is in a register or other memory location. The address may be an absolute or relative address. Compare *direct*.

IRB

The virtual-8086 mode interrupt-redirection bitmap.

IST

The long-mode interrupt-stack table.

IVT

The real-address mode interrupt-vector table.

LDT

Local descriptor table.

legacy x86

The legacy x86 architecture. See "Related Documents" on page xxiii for descriptions of the legacy x86 architecture.

legacy mode

An operating mode of the x86-64 architecture in which existing 16-bit and 32-bit applications and operating systems run without modification. A processor implementation of the x86-64 architecture can run in either *long mode* or *legacy mode*. Legacy mode has three submodes, *real mode*, *protected mode*, and *virtual-8086 mode*.

long mode

An operating mode unique to the x86-64 architecture. A processor implementation of the x86-64 architecture can run in either *long mode* or *legacy mode*. Long mode has two submodes, *64-bit mode* and *compatibility mode*.

lsb

Least-significant bit.

LSB

Least-significant byte.

main memory

Physical memory, such as RAM and ROM (but not cache memory) that is installed in a particular computer system.

mask

(1) A control bit that prevents the occurrence of a floating-point exception from invoking an exception-handling routine. (2) A field of bits used for a control purpose.

MBZ

Must be zero. If software attempts to set an MBZ bit to 1, a general-protection exception (#GP) occurs.

memory

Unless otherwise specified, *main memory.*

ModRM

A byte following an instruction opcode that specifies address calculation based on mode (Mod), register (R), and memory (M) variables.

moffset

A direct memory offset. In other words, a displacement that is added to the base of a code segment (for absolute addressing) or to an instruction pointer (for addressing relative to the instruction pointer, as in RIP-relative addressing).

msb

Most-significant bit.

MSB

Most-significant byte.

multimedia instructions

A combination of *128-bit media instructions* and *64-bit media instructions*.

octword

Same as *double quadword.*

offset

Same as *displacement*.

overflow

The condition in which a floating-point number is larger in magnitude than the largest, finite, positive or negative number that can be represented in the data-type format being used.

packed

See *vector.*

PAE

Physical-address extensions.

physical memory

Actual memory, consisting of *main memory* and cache.

probe

A check for an address in a processor's caches or internal buffers. *External probes* originate outside the processor, and *internal probes* originate within the processor.

protected mode

A submode of *legacy mode*.

quadword

Four words, or eight bytes, or 64 bits.

RAZ

Read as zero (0), regardless of what is written.

real-address mode

See *real mode*.

real mode

A short name for *real-address mode*, a submode of *legacy mode*.

relative

Referencing with a displacement (also called offset) from an instruction pointer rather than the base of a code segment. Contrast with *absolute*.

REX

An instruction prefix that specifies a 64-bit operand size and provides access to additional registers.

RIP-relative addressing

Addressing relative to the 64-bit RIP instruction pointer. Compare *moffset*.

set

To write a bit value of 1. Compare *clear*.

SIB

A byte following an instruction opcode that specifies address calculation based on scale (S), index (I), and base (B).

SIMD

Single instruction, multiple data. See *vector*.

SSE

Streaming SIMD extensions instruction set. See *128-bit media instructions* and *64-bit media instructions*.

SSE2

Extensions to the SSE instruction set. See *128-bit media instructions* and *64-bit media instructions.*

sticky bit

A bit that is set or cleared by hardware and that remains in that state until explicitly changed by software.

TOP

The x87 top-of-stack pointer.

TPR

Task-priority register (CR8).

TSS

Task-state segment.

underflow

The condition in which a floating-point number is smaller in magnitude than the smallest nonzero, positive or negative number that can be represented in the data-type format being used.

vector

(1) A set of integer or floating-point values, called *elements*, that are packed into a single operand. Most of the 128-bit and 64-bit media instructions use vectors as operands. Vectors are also called *packed* or *SIMD* (single-instruction multiple-data) operands.

(2) An index into an interrupt descriptor table (IDT), used to access exception handlers. Compare *exception*.

virtual-8086 mode

A submode of *legacy mode*.

word

Two bytes, or 16 bits.

x86

See *legacy x86*.

Registers

In the following list of registers, the names are used to refer either to a given register or to the contents of that register:

AH–DH

The high 8-bit AH, BH, CH, and DH registers. Compare *AL–DL*.

AL–DL

The low 8-bit AL, BL, CL, and DL registers. Compare *AH–DH.*

AL–r15B

The low 8-bit AL, BL, CL, DL, SIL, DIL, BPL, SPL, and R8B–R15B registers, available in 64-bit mode.

BP

Base pointer register.

CRn

Control register number *n*.

CS

Code segment register.

eAX–eSP

The 16-bit AX, BX, CX, DX, DI, SI, BP, and SP registers or the 32-bit EAX, EBX, ECX, EDX, EDI, ESI, EBP, and ESP registers. Compare *rAX–rSP.*

EBP

Extended base pointer register.

EFER

Extended features enable register.

eFLAGS

16-bit or 32-bit flags register. Compare *rFLAGS*.

EFLAGS
32-bit (extended) flags register.

eIP
16-bit or 32-bit instruction-pointer register. Compare *rIP*.

EIP
32-bit (extended) instruction-pointer register.

FLAGS
16-bit flags register.

GDTR
Global descriptor table register.

GPRs
General-purpose registers. For the 16-bit data size, these are AX, BX, CX, DX, DI, SI, BP, and SP. For the 32-bit data size, these are EAX, EBX, ECX, EDX, EDI, ESI, EBP, and ESP. For the 64-bit data size, these include RAX, RBX, RCX, RDX, RDI, RSI, RBP, RSP, and R8–R15.

IDTR
Interrupt descriptor table register.

IP
16-bit instruction-pointer register.

LDTR
Local descriptor table register.

MSR
Model-specific register.

r8–r15
The 8-bit R8B–R15B registers, or the 16-bit R8W–R15W registers, or the 32-bit R8D–R15D registers, or the 64-bit R8–R15 registers.

rAX–rSP
The 16-bit AX, BX, CX, DX, DI, SI, BP, and SP registers, or the 32-bit EAX, EBX, ECX, EDX, EDI, ESI, EBP, and ESP registers, or the 64-bit RAX, RBX, RCX, RDX, RDI, RSI, RBP, and RSP registers. Replace the placeholder *r* with

nothing for 16-bit size, "E" for 32-bit size, or "R" for 64-bit size.

RAX

64-bit version of the EAX register.

RBP

64-bit version of the EBP register.

RBX

64-bit version of the EBX register.

RCX

64-bit version of the ECX register.

RDI

64-bit version of the EDI register.

RDX

64-bit version of the EDX register.

rFLAGS

16-bit, 32-bit, or 64-bit flags register. Compare *RFLAGS*.

RFLAGS

64-bit flags register. Compare *rFLAGS*.

rIP

16-bit, 32-bit, or 64-bit instruction-pointer register. Compare *RIP*.

RIP

64-bit instruction-pointer register.

RSI

64-bit version of the ESI register.

RSP

64-bit version of the ESP register.

SP

Stack pointer register.

SS

Stack segment register.

TPR

Task priority register, a new register introduced in the x86-64 architecture to speed interrupt management.

TR

Task register.

Endian Order

The x86 and x86-64 architectures address memory using little-endian byte-ordering. Multibyte values are stored with their least-significant byte at the lowest byte address, and they are illustrated with their least significant byte at the right side. Strings are illustrated in reverse order, because the addresses of their bytes increase from right to left.

Related Documents

- Peter Abel, *IBM PC Assembly Language and Programming*, Prentice-Hall, Englewood Cliffs, NJ, 1995.
- Rakesh Agarwal, *80x86 Architecture & Programming: Volume II*, Prentice-Hall, Englewood Cliffs, NJ, 1991.
- AMD, *AMD-K6™ MMX™ Enhanced Processor Multimedia Technology*, Sunnyvale, CA, 2000.
- AMD, *3DNow!™ Technology Manual*, Sunnyvale, CA, 2000.
- AMD, *AMD Extensions to the 3DNow!™ and MMX™ Instruction Sets*, Sunnyvale, CA, 2000.
- Don Anderson and Tom Shanley, *Pentium Processor System Architecture*, Addison-Wesley, New York, 1995.
- Nabajyoti Barkakati and Randall Hyde, *Microsoft Macro Assembler Bible*, Sams, Carmel, Indiana, 1992.
- Barry B. Brey, *8086/8088, 80286, 80386, and 80486 Assembly Language Programming*, Macmillan Publishing Co., New York, 1994.
- Barry B. Brey, *Programming the 80286, 80386, 80486, and Pentium Based Personal Computer*, Prentice-Hall, Englewood Cliffs, NJ, 1995.
- Ralf Brown and Jim Kyle, *PC Interrupts,* Addison-Wesley, New York, 1994.
- Penn Brumm and Don Brumm, *80386/80486 Assembly Language Programming*, Windcrest McGraw-Hill, 1993.
- Geoff Chappell, *DOS Internals,* Addison-Wesley, New York, 1994.

- Chips and Technologies, Inc. *Super386 DX Programmer's Reference Manual*, Chips and Technologies, Inc., San Jose, 1992.
- John Crawford and Patrick Gelsinger, *Programming the 80386*, Sybex, San Francisco, 1987.
- Cyrix Corporation, *5x86 Processor BIOS Writer's Guide*, Cyrix Corporation, Richardson, TX, 1995.
- Cyrix Corporation, *M1 Processor Data Book*, Cyrix Corporation, Richardson, TX, 1996.
- Cyrix Corporation, *MX Processor MMX Extension Opcode Table*, Cyrix Corporation, Richardson, TX, 1996.
- Cyrix Corporation, *MX Processor Data Book*, Cyrix Corporation, Richardson, TX, 1997.
- Jeffrey P. Doyer, *Introduction to Protected Mode Programming*, course materials for an onsite class, 1992.
- Ray Duncan, *Extending DOS: A Programmer's Guide to Protected-Mode DOS*, Addison Wesley, NY, 1991.
- William B. Giles, *Assembly Language Programming for the Intel 80xxx Family*, Macmillan, New York, 1991.
- Frank van Gilluwe, *The Undocumented PC,* Addison-Wesley, New York, 1994.
- John L. Hennessy and David A. Patterson, *Computer Architecture*, Morgan Kaufmann Publishers, San Mateo, CA, 1996.
- Thom Hogan, *The Programmer's PC Sourcebook*, Microsoft Press, Redmond, WA, 1991.
- Hal Katircioglu, *Inside the 486, Pentium, and Pentium Pro*, Peer-to-Peer Communications, Menlo Park, CA, 1997.
- IBM Corporation, *486SLC Microprocessor Data Sheet*, IBM Corporation, Essex Junction, VT, 1993.
- IBM Corporation, *486SLC2 Microprocessor Data Sheet*, IBM Corporation, Essex Junction, VT, 1993.
- IBM Corporation, *80486DX2 Processor Floating Point Instructions*, IBM Corporation, Essex Junction, VT, 1995.
- IBM Corporation, *80486DX2 Processor BIOS Writer's Guide*, IBM Corporation, Essex Junction, VT, 1995.
- IBM Corporation, *Blue Lightening 486DX2 Data Book*, IBM Corporation, Essex Junction, VT, 1994.

- Institute of Electrical and Electronics Engineers, *IEEE Standard for Binary Floating-Point Arithmetic*, ANSI/IEEE Std 754-1985.
- Institute of Electrical and Electronics Engineers, *IEEE Standard for Radix-Independent Floating-Point Arithmetic*, ANSI/IEEE Std 854-1987.
- Muhammad Ali Mazidi and Janice Gillispie Mazidi, *80X86 IBM PC and Compatible Computers*, Prentice-Hall, Englewood Cliffs, NJ, 1997.
- Hans-Peter Messmer, *The Indispensable Pentium Book,* Addison-Wesley, New York, 1995.
- Karen Miller, *An Assembly Language Introduction to Computer Architecture: Using the Intel Pentium*, Oxford University Press, New York, 1999.
- Stephen Morse, Eric Isaacson, and Douglas Albert, *The 80386/387 Architecture*, John Wiley & Sons, New York, 1987.
- NexGen Inc., *Nx586 Processor Data Book*, NexGen Inc., Milpitas, CA, 1993.
- NexGen Inc., *Nx686 Processor Data Book*, NexGen Inc., Milpitas, CA, 1994.
- Bipin Patwardhan, *Introduction to the Streaming SIMD Extensions in the Pentium III*, www.x86.org/articles/sse_pt1/simd1.htm, June, 2000.
- Peter Norton, Peter Aitken, and Richard Wilton, *PC Programmer's Bible,* Microsoft Press, Redmond, WA, 1993.
- *PharLap 386|ASM Reference Manual*, Pharlap, Cambridge MA, 1993.
- *PharLap TNT DOS-Extender Reference Manual*, Pharlap, Cambridge MA, 1995.
- Sen-Cuo Ro and Sheau-Chuen Her, *i386/i486 Advanced Programming*, Van Nostrand Reinhold, New York, 1993.
- Tom Shanley, *Protected Mode System Architecture*, Addison Wesley, NY, 1996.
- SGS-Thomson Corporation, *80486DX Processor SMM Programming Manual*, SGS-Thomson Corporation, 1995.
- Walter A. Triebel, *The 80386DX Microprocessor*, Prentice-Hall, Englewood Cliffs, NJ, 1992.
- John Wharton, *The Complete x86*, MicroDesign Resources, Sebastopol, California, 1994.

- Web sites and newsgroups:
  - www.amd.com
  - news.comp.arch
  - news.comp.lang.asm.x86
  - news.intel.microprocessors
  - news.microsoft

1 64-Bit Media Instruction Reference

This chapter describes the function, mnemonic syntax, opcodes, affected flags, and possible exceptions generated by the 64-bit media instructions. These instructions operate on data located in the 64-bit MMX™ registers. Most of the instructions operate in parallel on sets of packed elements called *vectors*, although some operate on scalars. The instructions define both integer and floating-point operations, and include the legacy MMX instructions, the 3DNow!™ instructions, and the AMD extensions to the MMX and 3DNow! instruction sets.

Each instruction that performs a vector (packed) operation is illustrated with a diagram. Figure 1-1 on page 2 shows the conventions used in these diagrams. The particular diagram shows the PSLLW (packed shift left logical words) instruction.

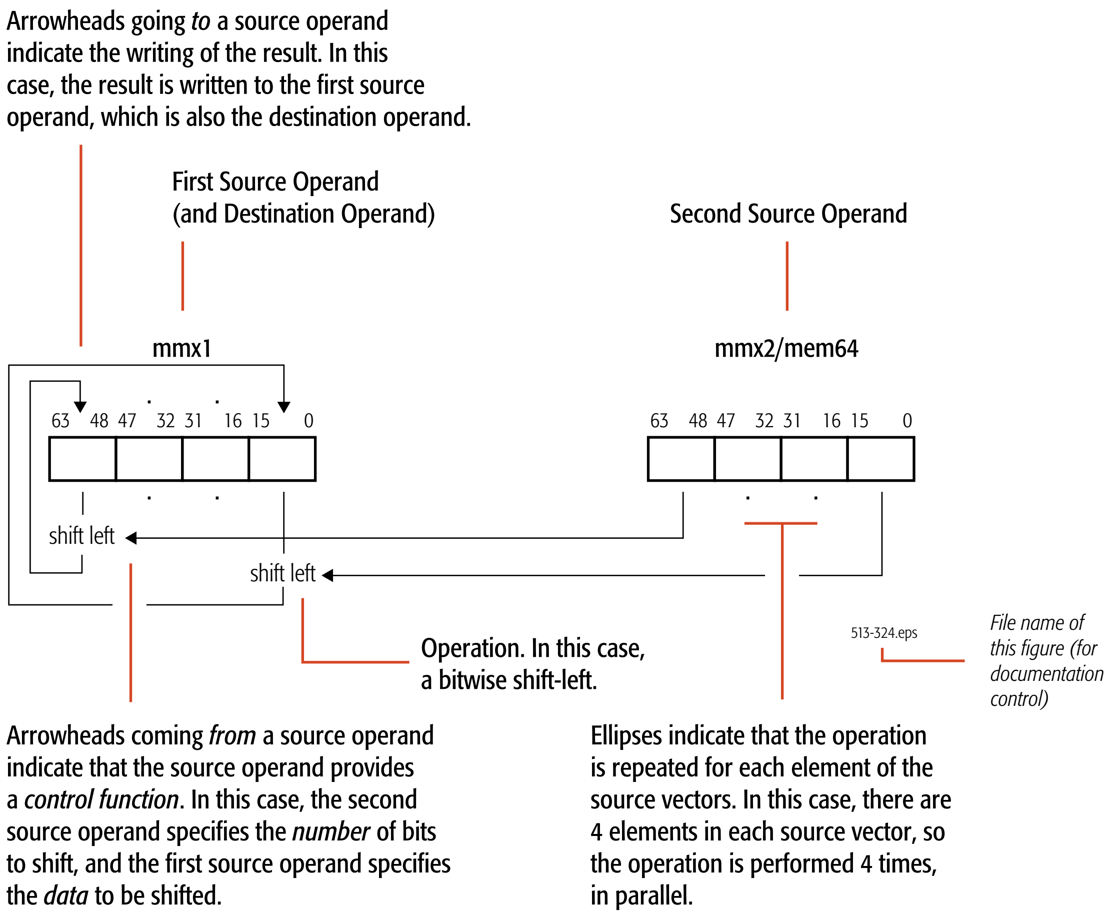


Figure 1-1. Diagram Conventions for 64-Bit Media Instructions

Gray areas in diagrams indicate unmodified operand bits.

Like the 128-bit media instructions, many of the 64-bit instructions independently and simultaneously perform a single operation on multiple elements of a vector and are thus classified as *single-instruction, multiple-data* (SIMD) instructions. A few 64-bit media instructions convert operands in MMX registers to operands in GPR, XMM, or x87 registers (or vice versa), or save or restore MMX state, or reset x87state.

Hardware support for a specific 64-bit media instruction depends on the presence of at least one of the following CPUID functions:

- MMX Instructions, indicated by bit 23 of CPUID standard function 1 and extended function 8000_0001h.

- AMD Extensions to MMX Instructions, indicated by bit 22 of CPUID extended function 8000_0001h.
- SSE, indicated by bit 25 of CPUID extended function 8000_0001h.
- AMD 3DNow! Instructions, indicated by bit 31 of CPUID extended function 8000_0001h.
- AMD Extensions to 3DNow! Instructions, indicated by bit 30 of CPUID extended function 8000_0001h.
- FXSAVE and FXRSTOR, indicated by bit 24 of CPUID standard function 1 and extended function 8000_0001h.

The 64-bit media instructions can be used in legacy mode or long mode. Their use in long mode is available if the following CPUID function is set:

- Long Mode, indicated by bit 29 of CPUID extended function 8000_0001h.

Compilation of 64-bit media programs for execution in 64-bit mode offers four primary advantages: access to the eight extended, 64-bit general-purpose registers (for a register set consisting of GPR0–GPR15), access to the eight extended XMM registers (for a register set consisting of XMM0–XMM15), access to the 64-bit virtual address space, and access to the RIP-relative addressing mode.

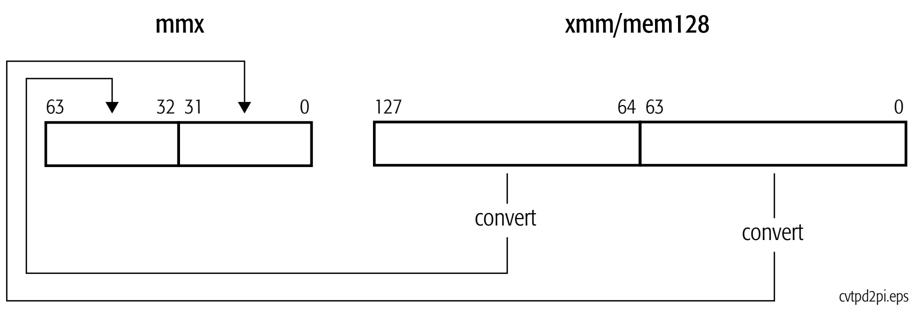
For further information, see:

- "64-Bit Media Programming" in volume 1.
- "Summary of Registers and Data Types" in volume 3.
- "Notation" in volume 3.
- "Instruction Prefixes" in volume 3.

CVTPD2PI Convert Packed Double-Precision Floating-Point to Packed Doubleword Integers

The CVTPD2PI instruction converts two packed double-precision floating-point values in an XMM register or a 128-bit memory location to two packed 32-bit signed integer values and writes the converted values in an MMX register.

Mnemonic	Opcode	Description
CVTPD2PI *mmx, xmm2/mem128*	66 0F 2D /*r*	Converts packed double-precision floating-point values in an XMM register or 128-bit memory location to packed doubleword integers values in the destination MMX™ register.



If the result of the conversion is an inexact value, the value is rounded as specified by the rounding control bits (RC) in the MXCSR register. If the floating-point value is a NaN, infinity, or if the result of the conversion is larger than the maximum signed doubleword (-2^{31} to $+2^{31} - 1$), the instruction returns the 32-bit indefinite integer value (8000_0000h) when the invalid-operation exception (IE) is masked.

Execution of this instruction causes all fields in the x87 tag word to be set according to their corresponding data, the top-of-stack-pointer bit (TOP) in the x87 status word to be cleared to 0, and any pending x87 exceptions are handled before this instruction is executed. For details, see "Actions Taken on Executing 64-Bit Media Instructions" in volume 1.

Related Instructions

CVTDQ2PD, CVTPD2DQ, CVTPI2PD, CVTSD2SI, CVTSI2SD, CVTTPD2DQ, CVTTPD2PI, CVTTSD2SI

rFLAGS Affected

None

MXCSR Flags Affected

	FZ	RC		PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
											M					M
31 16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note:
A flag that can be set to one or zero is M (modified). Unaffected flags are blank. Shaded fields are reserved.

Exceptions

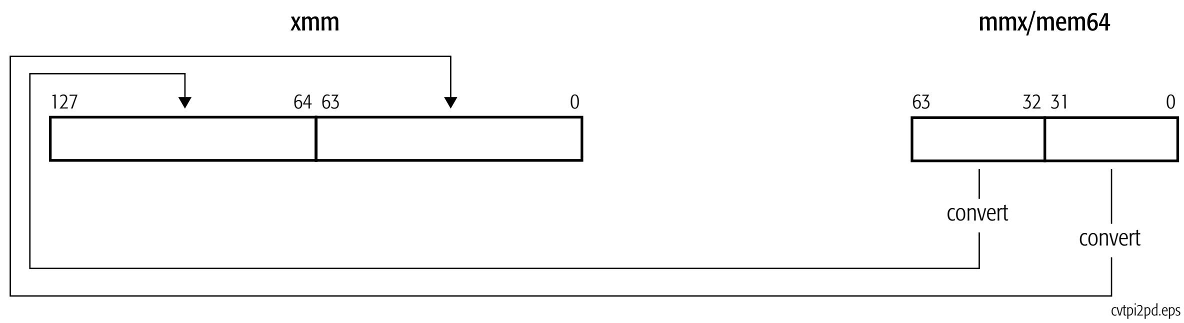
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 was cleared to 0 and there was an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 was cleared to 0. The SSE2 instructions are not supported, as indicated by bit 26 in CPUID extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	The memory operand was not aligned on a 16-byte boundary.
	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 was set to 1, and there was an unmasked SIMD floating-point exception. See *SIMD Floating-Point Exceptions*, below, for details.

Exception	Real	Virtual 8086	Protected	Cause of Exception
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	The source operand was an SNaN, QNaN, or infinity, or there is an overflow.
Precision exception (PE)	X	X	X	The result could not be represented exactly in the destination format.

CVTPI2PD Convert Packed Doubleword Integers to Packed Double-Precision Floating-Point

The CVTPI2PD instruction converts two packed 32-bit signed integer values in an MMX register or a 64-bit memory location to two double-precision floating-point values and writes the converted values in an XMM register.

Mnemonic	Opcode	Description
CVTPI2PD *xmm*, *mmx/mem64*	66 0F 2A /r	Converts two packed doubleword integer values in an MMX™ register or 64-bit memory location to two packed double-precision floating-point values in the destination XMM register.



Execution of this instruction causes all fields in the x87 tag word to be set according to their corresponding data, the top-of-stack-pointer bit (TOP) in the x87 status word to be cleared to 0, and any pending x87 exceptions are handled before this instruction is executed. For details, see "Actions Taken on Executing 64-Bit Media Instructions" in volume 1.

Related Instructions

CVTDQ2PD, CVTPD2DQ, CVTPD2PI, CVTSD2SI, CVTSI2SD, CVTTPD2DQ, CVTTPD2PI, CVTTSD2SI

rFLAGS Affected

None

MXCSR Flags Affected

None

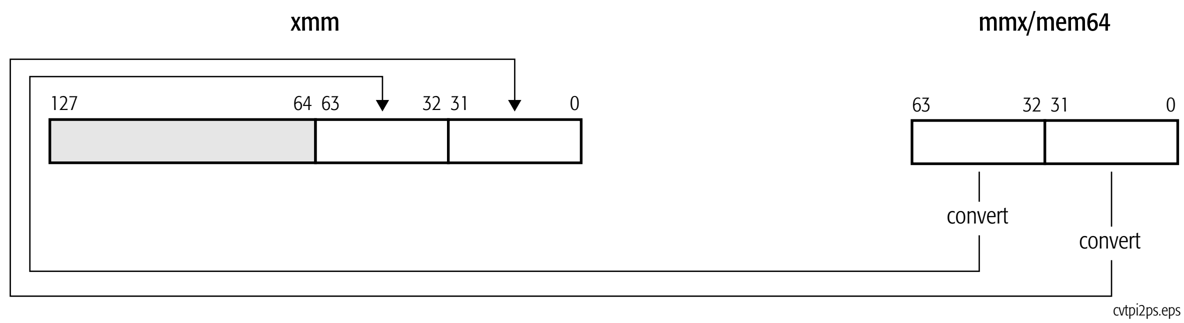
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 was cleared to 0 and there was an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 was cleared to 0. The SSE2 instructions are not supported, as indicated by bit 26 in CPUID extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking is enabled.
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 was set to 1, and there was an unmasked SIMD floating-point exception. See *SIMD Floating-Point Exceptions*, below, for details.
SIMD Floating-Point Exceptions				
Precision exception (PE)	X	X	X	The result could not be represented exactly in the destination format.

CVTPI2PS Convert Packed Doubleword Integers to Packed Single-Precision Floating-Point

The CVTPI2PS instruction converts two packed 32-bit signed integer values in an MMX register or a 64-bit memory location to two single-precision floating-point values and writes the converted values in the low-order 64 bits of an XMM register. The high-order 64 bits of the XMM register are not modified.

Mnemonic	Opcode	Description
CVTPI2PS *xmm*, *mmx/mem64*	0F 2A /*r*	Converts packed doubleword integer values in an MMX™ register or 64-bit memory location to single-precision floating-point values in the destination XMM register.



Execution of this instruction causes all fields in the x87 tag word to be set according to their corresponding data, the top-of-stack-pointer bit (TOP) in the x87 status word to be cleared to 0, and any pending x87 exceptions are handled before this instruction is executed. For details, see “Actions Taken on Executing 64-Bit Media Instructions” in volume 1.

Related Instructions

CVTDQ2PS, CVTPS2DQ, CVTPS2PI, CVTSI2SS, CVTSS2SI, CVTTPS2DQ, CVTTPS2PI, CVTTSS2SI

rFLAGS Affected

None

MXCSR Flags Affected

	FZ	RC		PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
											M					
31 16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note:
A flag that can be set to one or zero is M (modified). Unaffected flags are blank. Shaded fields are reserved.

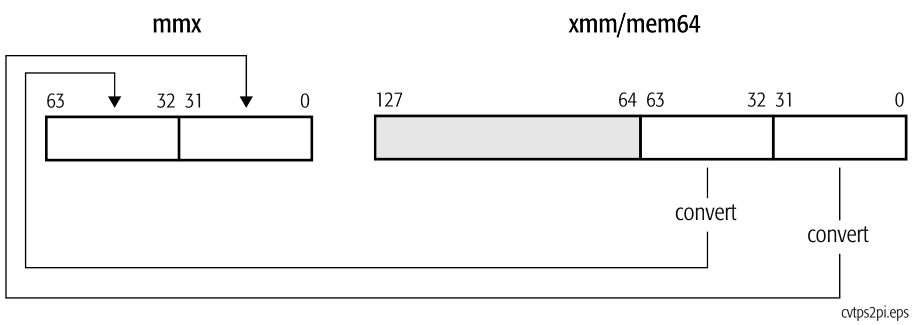
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 was cleared to 0 and there was an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 was cleared to 0. The SSE2 instructions are not supported, as indicated by bit 26 in CPUID extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeds the stack segment limit or is non-canonical.
General protection, #GP	X	X	X	A memory address exceeds a data segment limit or is non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking is enabled.
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See *SIMD Floating-Point Exceptions*, below, for details.
SIMD Floating-Point Exceptions				
Precision exception (PE)	X	X	X	The result could not be represented exactly in the destination format.

CVTPS2PI — Convert Packed Single-Precision Floating-Point to Packed Doubleword Integers

The CVTPS2PI instruction converts two packed single-precision floating-point values in the low-order 64 bits of an XMM register or a 64-bit memory location to two packed 32-bit signed integers and writes the converted values in an MMX register.

Mnemonic	Opcode	Description
CVTPS2PI *mmx, xmm/mem64*	0F 2D /*r*	Converts packed single-precision floating-point values in an XMM register or 64-bit memory location to packed doubleword integers in the destination MMX™ register.



If the result of the conversion is an inexact value, the value is rounded as specified by the rounding control bits (RC) in the MXCSR register. If the floating-point value is a NaN, infinity, or if the result of the conversion is larger than the maximum signed doubleword (-2^{31} to $+2^{31}-1$), the instruction returns the 32-bit indefinite integer value (8000_0000h) when the invalid-operation exception (IE) is masked.

Execution of this instruction causes all fields in the x87 tag word and the top-of-stack-pointer bit (TOP) in the x87 status word to be cleared to 0, and any pending x87 exceptions are handled before this instruction is executed. For details, see “Actions Taken on Executing 64-Bit Media Instructions” in volume 1.

Related Instructions

CVTDQ2PS, CVTPI2PS, CVTPS2DQ, CVTSI2SS, CVTSS2SI, CVTTPS2DQ, CVTTPS2PI, CVTTSS2SI

rFLAGS Affected

None

MXCSR Flags Affected

	FZ	RC		PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
											M					M
31 16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note:
A flag that can be set to one or zero is M (modified). Unaffected flags are blank. Shaded fields are reserved.

Exceptions

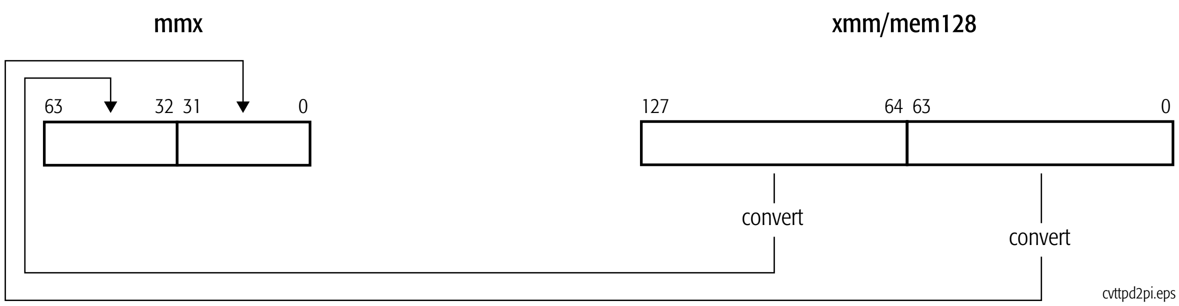
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 was cleared to 0 and there was an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 was cleared to 0. The SSE2 instructions are not supported, as indicated by bit 26 in CPUID extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking is enabled.
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See *SIMD Floating-Point Exceptions*, below, for details.

Exception	Real	Virtual 8086	Protected	Cause of Exception
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	The source operand was an SNaN, QNaN, or infinity, or there was an overflow.
Precision exception (PE)	X	X	X	The result could not be represented exactly in the destination format.

CVTTPD2PI — Convert Packed Double-Precision Floating-Point to Packed Doubleword Integers, Truncated

The CVTTPD2PI instruction converts two packed double-precision floating-point values in an XMM register or a 128-bit memory location to two packed 32-bit signed integer values and writes the converted values in an MMX register.

Mnemonic	Opcode	Description
CVTPD2PI *mmx, xmm/mem128*	66 0F 2C /*r*	Converts packed double-precision floating-point values in an XMM register or 128-bit memory location to packed doubleword integer values in the destination MMX™ register. Inexact results are truncated.



If the result of the conversion is an inexact value, the value is truncated (rounded toward zero). If the floating-point value is a NaN, infinity, or if the result of the conversion is larger than the maximum signed doubleword (-2^{31} to $+2^{31}-1$), the instruction returns the 32-bit indefinite integer value (8000_0000h) when the invalid-operation exception (IE) is masked.

Execution of this instruction causes all fields in the x87 tag word to be set according to their corresponding data, the top-of-stack-pointer bit (TOP) in the x87 status word to be cleared to 0, and any pending x87 exceptions are handled before this instruction is executed. For details, see “Actions Taken on Executing 64-Bit Media Instructions” in volume 1.

Related Instructions

CVTDQ2PD, CVTPD2DQ, CVTPD2PI, CVTPI2PD, CVTSD2SI, CVTSI2SD, CVTTPD2DQ, CVTTSD2SI

rFLAGS Affected

None

MXCSR Flags Affected

	FZ	RC		PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
											M					M
31 16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note:
A flag that can be set to one or zero is M (modified). Unaffected flags are blank. Shaded fields are reserved.

Exceptions

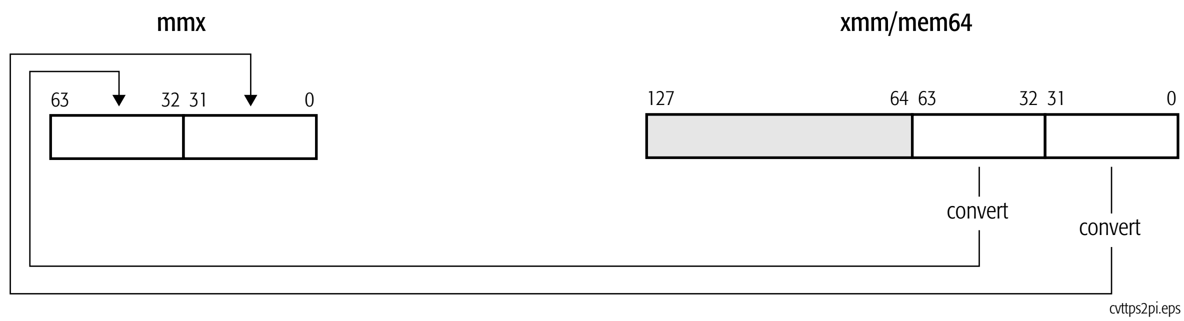
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 was cleared to 0 and there was an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 was cleared to 0. The SSE2 instructions are not supported, as indicated by bit 26 in CPUID extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	The memory operand was not aligned on a 16-byte boundary.
			X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See *SIMD Floating-Point Exceptions*, below, for details.

Exception	Real	Virtual 8086	Protected	Cause of Exception
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	The source operand was an SNaN, QNaN, or infinity, or there was an overflow.
Precision exception (PE)	X	X	X	The result could not be represented exactly in the destination format.

CVTTPS2PI Convert Packed Single-Precision Floating-Point to Packed Doubleword Integers, Truncated

The CVTTPS2PI instruction converts two packed single-precision floating-point values in the low-order 64 bits of an XMM register or a 64-bit memory location to two packed 32-bit signed integer values and writes the converted values in an MMX register.

Mnemonic	Opcode	Description
CVTTPS2PI *mmx xmm/mem64*	0F 2C /*r*	Converts packed single-precision floating-point values in an XMM register or 64-bit memory location to doubleword integer values in the destination MMX™ register. Inexact results are truncated.



If the result of the conversion is an inexact value, the value is truncated (rounded toward zero). If the floating-point value is a NaN, infinity, or if the result of the conversion is larger than the maximum signed doubleword (-2^{31} to $+2^{31} - 1$), the instruction returns the 32-bit indefinite integer value (8000_0000h) when the invalid-operation exception (IE) is masked.

Execution of this instruction causes all fields in the x87 tag word to be set according to their corresponding data, the top-of-stack-pointer bit (TOP) in the x87 status word to be cleared to 0, and any pending x87 exceptions are handled before this instruction is executed. For details, see "Actions Taken on Executing 64-Bit Media Instructions" in volume 1.

Related Instructions

CVTDQ2PS, CVTPI2PS, CVTPS2DQ, CVTPS2PI, CVTSI2SS, CVTSS2SI, CVTTPS2DQ, CVTTSS2SI

rFLAGS Affected

None

MXCSR Flags Affected

	FZ	RC		PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
											M					M
31 16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note:
A flag that can be set to one or zero is M (modified). Unaffected flags are blank. Shaded fields are reserved.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 was cleared to 0 and there was an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 was cleared to 0. The SSE2 instructions are not supported, as indicated by bit 26 in CPUID extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking is enabled.
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See *SIMD Floating-Point Exceptions*, below, for details.

Exception	Real	Virtual 8086	Protected	Cause of Exception
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	The source operand was an SNaN, QNaN, or infinity, or there was an overflow.
Precision exception (PE)	X	X	X	The result could not be represented exactly in the destination format.

EMMS — Exit Multimedia State

The EMMS instruction clears the MMX state by setting the state of the x87 stack registers to *empty* (tag-bit encoding of all 1s for all MMX registers) indicating that the contents of the registers are available for a new procedure, such as an x87 floating-point procedure. This setting of the tag bits is referred to as "clearing the MMX state".

Because the MMX registers and tag word are shared with the x87 floating-point instructions, software should execute an EMMS instruction to clear the MMX state before executing code that includes x87 floating-point instructions.

The functions of the EMMS and FEMMS instructions are identical.

For details about the setting of x87 tag bits, see "Media and x87 Processor State" in volume 2.

Mnemonic	Opcode	Description
EMMS	0F 77	Clears the MMX™ state.

Related Instructions

FEMMS (a 3DNow! instruction)

rFLAGS Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The MMX™ instructions are not supported, as indicated by bit 23 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.

FEMMS Fast Exit Multimedia State

The FEMMS instruction clears the MMX state by setting the state of the x87 stack registers to *empty* (tag-bit encoding of all 1s for all MMX registers) indicating that the contents of the registers are available for a new procedure, such as an x87 floating-point procedure. This setting of the tag bits is referred to as “clearing the MMX state”.

Because the MMX registers and tag word are shared with the x87 floating-point instructions, software should execute an EMMS or FEMMS instruction to clear the MMX state before executing code that includes x87 floating-point instructions.

FEMMS is a 3DNow! instruction. The functions of the FEMMS and EMMS instructions are identical. The FEMMS instruction is supported for backward-compatibility with certain AMD processors. Software that must be both compatible with both AMD and non-AMD processors should use the EMMS instruction.

For details about the setting of x87 tag bits, see “Media and x87 Processor State” in volume 2.

Mnemonic	Opcode	Description
FEMMS	0F 0E	Clears MMX™ state.

Related Instructions

EMMS

rFLAGS Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The AMD 3DNow!™ instructions are not supported, as indicated by bit 31 in CPUID extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.

FNSAVE Save No-Wait x87 and MMX™ State

The FNSAVE instruction writes the complete x87 state to a 94-byte or 108-byte area in memory, depending upon whether the processor is operating in real or protected mode and whether the operand-size attribute is 16-bit or 32-bit, and reinitializes the x87 state. The MMX state is also saved by virtue of the fact that MMX state shares part of the x87 state. The x87 condition codes are cleared after being saved.

Unlike FSAVE, FNSAVE does not wait for pending unmasked x87 floating-point exceptions to occur. This no-wait instruction should be used when processor interrupts are disabled and x87 instructions might generate an interrupt that would create an endless wait.

For details about the memory image saved by FNSAVE, see "Media and x87 Processor State" in volume 2.

Mnemonic	Opcode	Description
FNSAVE *mem94/108env*	DD /6	Store a copy of the floating-pointing state to *mem94/108env* without checking for pending floating-point exceptions, then reinitialize the x87 state.

Examples

1. Save environment and stack data:

```
FNSAVE memory94Ptr      ;Save the x87 environment.
.
.     ;Do something in the clean operating environment.
.
FRSTOR memory94Ptr      ;Restore the old operating environment.
```

Related Instructions

FSAVE, FRSTOR, FXSAVE, FXRSTOR

x87 Condition Code

x87 Condition Code	Description
C0	0
C1	0
C2	0
C3	0

Exceptions (All Modes)

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) was set to 1.
Stack, #SS	X	X	X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	Result was located in a nonwritable segment.
			X	A null data segment was used to reference memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking is enabled.

FRSTOR Restore x87 and MMX™ State

The FRSTOR instruction restores complete x87 state from a 94-byte or 108-byte area in memory, depending upon whether the processor is operating in real or protected mode and whether the operand-size attribute is 16-bit or 32-bit. The MMX state is also restored by virtue of the fact that MMX state shares part of the x87 state. The x87 condition codes are loaded from memory.

Unlike the FXRSTOR instruction, FRSTOR checks for pending unmasked x87 floating-point exceptions in the restored image. However, if such a pending unmasked exception is found, the exception does not occur immediately after execution of an FRSTOR instruction. Instead, the processor asserts the FERR# (floating-point error) output signal. The normal rules for x87 exceptions then apply, just as if the exception was created by an x87 instruction. If CR0.NE = 1, the processor vectors to the x87 floating-point exception (#MF) handler at the next non-wait x87 instruction. Otherwise, the processor examines the state of the IGNNE# (ignore numeric error) input signal. If IGNNE# is asserted, the processor executes the next x87 instruction. If IGNNE# is negated, the processor freezes at that instruction boundary until an external interrupt occurs that redirects control to a handler that clears the exception.

For details about the memory image restored by FRSTOR, see “Media and x87 Processor State” in volume 2.

Mnemonic	Opcode	Description
FRSTOR *mem94/108env*	DD /4	Load a copy of the floating-pointing state from *mem94/108env*.

Examples

1. Save environment and stack data:

```
FSAVE memory94Ptr        ;Save the x87 environment.
.
.     ;Do something in the clean operating environment.
.
FRSTOR memory94Ptr       ;Restore the old operating environment.
```

Related Instructions

FSAVE, FNSAVE, FXSAVE, FXRSTOR

x87 Condition Code

x87 Condition Code	Description
C0	Loaded from memory.
C1	Loaded from memory.
C2	Loaded from memory.
C3	Loaded from memory.

Exceptions (All Modes)

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) was set to 1.
Stack, #SS	X	X	X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking is enabled.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.

FSAVE Save x87 and MMX™ State

The FSAVE instruction writes the complete x87 state to a 94-byte or 108-byte area in memory, depending upon whether the processor is operating in real or protected mode and whether the operand-size attribute is 16-bit or 32-bit, and initializes the x87 state. The MMX state is also saved by virtue of the fact that MMX state shares part of the x87 state. The x87 condition codes are cleared after being saved.

Assemblers issue FSAVE as an FWAIT instruction (9Bh) followed by an FNSAVE instruction. Thus, FSAVE (but not FNSAVE) reports pending unmasked x87 floating-point exceptions before saving the state.

For details about the memory image saved by FSAVE, see "Media and x87 Processor State" in volume 2.

Mnemonic	Opcode	Description
FSAVE *mem94/108env*	9B DD /6	Store a copy of the floating-pointing state to *mem94/108env* after checking for pending floating-point exceptions, then reinitialize the x87 state.

Examples

1. Save environment and stack data:

```
FSAVE memory94Ptr       ;Save the x87 environment.
.
.     ;Do something in the clean operating environment.
.
FRSTOR memory94Ptr      ;Restore the old operating environment.
```

Related Instructions

FNSAVE, FRSTOR, FXSAVE, FXRSTOR

x87 Condition Code

x87 Condition Code	Value	Description
C0	0	
C1	0	
C2	0	
C3	0	
U indicates 'undefined.'		

Exceptions (All Modes)

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) was set to 1.
Stack, #SS	X	X	X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	Result was located in a nonwritable segment.
			X	A null data segment was used to reference memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking is enabled.

FXRSTOR Restore XMM, MMX™, and x87 State

The FXRSTOR instruction restores the XMM, MMX, and x87 state. The data loaded from memory is the state information previously saved using the FXSAVE instruction. Restoring data with FXRSTOR that had been previously saved with an FSAVE (rather than FXSAVE) instruction results in an incorrect restoration.

.

Mnemonic	Opcode	Description
FXRSTOR *mem512env*	0F AE /1	Restores XMM, MMX™, and x87 state.

Unlike the FRSTOR instruction, FXRSTOR does not cause pending unmasked x87 floating-point exceptions in the restored image to be recognized when internal control of x87 exceptions is enabled (CR0.NE = 1). When loading a new environment, use an FWAIT instruction after the FXRSTOR instruction to check for and handle any pending unmasked x87 exceptions.

If the restored MXCSR register contains a set bit in an exception status flag, and the corresponding exception mask bit is cleared (indicating an unmasked exception), loading the MXCSR register from memory does not cause a SIMD floating-point exception (#XM).

FXRSTOR executes faster than FRSTOR because it does not restore the x87 error pointers (last instruction pointer, last data pointer, and last opcode) except in the relatively rare cases in which the exception-summary (ES) bit in the x87 status word is set to 1, indicating that an unmasked x87 exception has occurred.

The architecture supports two memory formats for FXRSTOR, a 512-byte 32-bit legacy format and a 512-byte 64-bit format. Selection of the 32-bit or 64-bit format is accomplished by using the corresponding effective operand size in the FXRSTOR instruction. If software running in 64-bit mode executes an FXRSTOR with a 32-bit operand size (no REX-prefix operand-size override), the 32-bit legacy format is used. If software running in 64-bit mode executes an FXRSTOR with a 64-bit operand size (requires REX-prefix operand-size override), the 64-bit format is used. For details about the memory image restored by FXRSTOR, see “Saving Media and x87 Processor State” in volume 2.

If the operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0, the saved image of XMM0–XMM7 and MXCSR is not loaded into the processor. A general-protection exception occurs if there is an attempt to load a non-zero value to the bits in MXCSR that are defined as reserved and cleared (bits 31–16 and bit 6).

Related Instructions

FWAIT, FXSAVE

rFLAGS Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The instruction was preceded by the LOCK prefix. The emulate bit (EM) of CR0 was set to 1. The FXSAVE/FSRSTOR instructions are not supported, as indicated by bit 24 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or is non-canonical.)
General protection, #GP	X	X	X	The memory operand was not aligned on a 16-byte boundary.
	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
			X	An attempt was made to write a non-zero value to reserved bits in MXCSR.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking is enabled.

FXSAVE Save XMM, MMX™, and x87 State

The FXSAVE instruction saves the XMM, MMX, and x87 state. A memory location that is not aligned on a 16-byte boundary causes a general-protection exception, or, in some cases, an alignment-check exception.

Mnemonic	Opcode	Description
FXSAVE *mem512env*	0F AE /0	Saves XMM, MMX™, and x87 state in 512-byte memory location.

Unlike FSAVE and FNSAVE, FXSAVE does not alter the x87 tag bits. The contents of the saved MMX/x87 data registers are retained, thus indicating that the registers may be valid (or whatever other value the x87 tag bits indicated prior to the save). To invalidate the contents of the MMX/x87 data registers after FXSAVE, software must execute an FINIT instruction. Also, FXSAVE (like FNSAVE) does not check for pending unmasked x87 floating-point exceptions. An FWAIT instruction can be used for this purpose.

FXSAVE executes faster than FSAVE or FNSAVE because it does not save the x87 pointer registers (last instruction pointer, last data pointer, and last opcode) except in the relatively rare cases in which the exception-summary (ES) bit in the x87 status word is set to 1, indicating that an unmasked x87 exception has occurred.

The architecture supports two memory formats for FXSAVE, a 512-byte 32-bit legacy format and a 512-byte 64-bit format. Selection of the 32-bit or 64-bit format is accomplished by using the corresponding effective operand size in the FXSAVE instruction. If software running in 64-bit mode executes an FXSAVE with a 32-bit operand size (no REX-prefix operand-size override), the 32-bit legacy format is used. If software running in 64-bit mode executes an FXSAVE with a 64-bit operand size (requires REX-prefix operand-size override), the 64-bit format is used. For details about the memory image restored by FXRSTOR, see "Saving Media and x87 Processor State" in volume 2.

If the operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0, FXSAVE does not save the image of XMM0–XMM7 and MXCSR. For details about the CR4.OSFXSR bit, see "FXSAVE/FXRSTOR Support (OSFXSR) Bit" in volume 2.

Related Instructions

FINIT, FNSAVE, FRSTOR, FSAVE, FXRSTOR, LDMXCSR, STMXCSR

rFLAGS Affected

None

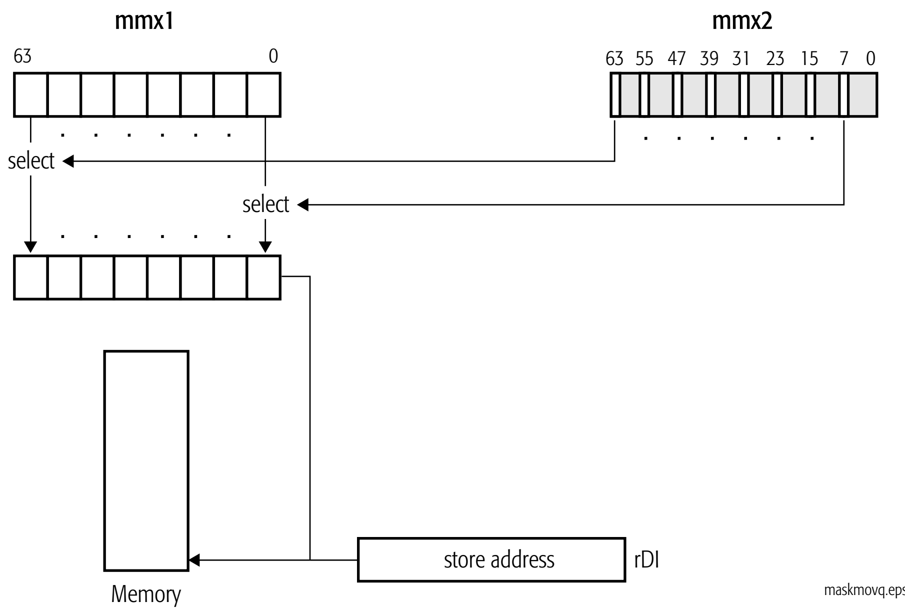
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The instruction was preceded by the LOCK prefix. The emulate bit (EM) of CR0 was set to 1. The FXSAVE/FSRSTOR instructions are not supported, as indicated by bit 24 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
			X	An attempt is made to write a non-zero value to reserved bits in MXCSR.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking is enabled.

MASKMOVQ Masked Move Quadword

The MASKMOVQ instruction stores bytes from the first source operand, as selected by the second source operand, to a memory location specified in the rDI and DS registers (except that DS is ignored in 64-bit mode). The first source operand is an MMX register, and the second source operand is another MMX register. The most-significant bit (msb) of each byte in the second source operand specifies the store (1 = store, 0 = no store) of the corresponding byte of the first source operand. The size of the store is determined by the address-size attribute.

Mnemonic	Opcode	Description
MASKMOVQ *mmx1*, *mmx2*	0F F7 /*r*	Store bytes from an MMX™ register, selected by the most-significant bit of the corresponding byte in another MMX™ register, to a memory location.



A mask value of all 0s results in the following behavior:

- No data is written to memory.
- Data breakpoints are not guaranteed to be signalled in all implementations (although code breakpoints are guaranteed).
- Page faults and some exceptions associated with memory addressing and are not

guaranteed to be signalled in all implementations.

- Illegal addresses in segment registers are signalled, as are segment overrun exceptions in real and virtual-86 modes.

MASKMOVQ implicitly uses weakly-ordered, write-combining buffering for the data, as described in “Buffering and Combining Memory Writes” in volume 2. If the stored data is shared by multiple processors, this instruction should be used together with a fence instruction in order to ensure data coherency (refer to “Cache and TLB Management” in volume 2).

Related Instructions

MASKMOVDQU

Exceptions

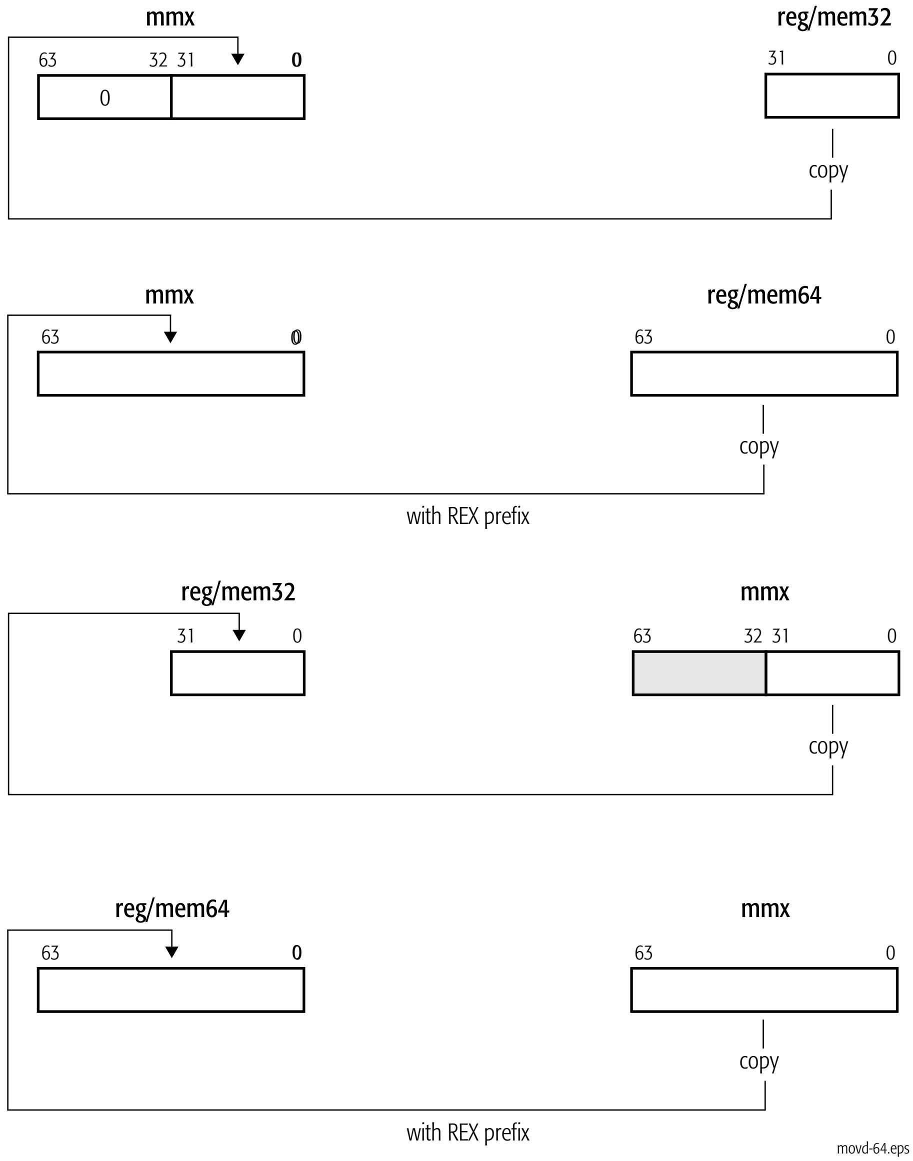
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 was cleared to 0. The SSE instructions are not supported, as indicated by bit 25 in CPUID extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking is enabled.

MOVD Move Doubleword or Quadword

The MOVD instruction moves a 32-bit or 64-bit value:

- from a 32-bit or 64-bit general-purpose register or memory location to the low-order 32 bits (with zero-extension to 64 bits) or the full 64 bits of an MMX register, or
- from the low-order 32 or the full 64 bits of an MMX register to a 32-bit or 64-bit general-purpose register or memory location.

Mnemonic	Opcode	Description
MOVD *mmx*, *reg/mem32*	0F 6E /*r*	Moves 32-bit value from a general-purpose register or 32-bit memory location to an MMX™ register.
MOVD *mmx*, *reg/mem64*	0F 6E /*r*	Moves 64-bit value from a general-purpose register or 64-bit memory location to an MMX™ register.
MOVD *reg/mem32*, *mmx*	0F 7E /*r*	Moves 32-bit value from an MMX™ register to a general-purpose register or 32-bit memory location.
MOVD *reg/mem64*, *mmx*	0F 7E /*r*	Moves 64-bit value from an MMX™ register to a general-purpose register or 64-bit memory location.



Related Instructions

MOVDQA, MOVDQU, MOVDQ2Q, MOVQ, MOVQ2DQ

rFLAGS Affected

None

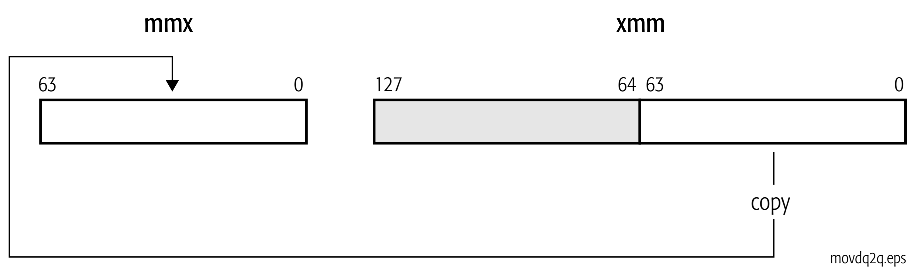
Exceptions

Exception	Real	Virtual 8086	Protected	Description
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The MMX instructions are not supported, as indicated by bit 23 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeds the stack segment limit or is non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking is enabled.

MOVDQ2Q Move Quadword to Quadword

The MOVDQ2Q instruction moves the low-order 64-bit value in an XMM register to a 64-bit MMX register.

Mnemonic	Opcode	Description
MOVDQ2Q *mmx*, *xmm*	F2 0F D6	Moves low-order 64-bit value from an XMM register to the destination MMX™ register.



Execution of this instruction causes all fields in the x87 tag word to be set according to their corresponding data, the top-of-stack-pointer bit (TOP) in the x87 status word to be cleared to 0, and any pending x87 exceptions are handled before this instruction is executed. For details, see "Actions Taken on Executing 64-Bit Media Instructions" in volume 1.

Related Instructions

MOVD, MOVDQA, MOVDQU, MOVQ, MOVQ2DQ

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The SSE2 instructions are not supported, as indicated by bit 26 in CPUID extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.

MOVNTQ Move Non-Temporal Quadword

The MOVNTQ instruction stores a 64-bit MMX register value into a 64-bit memory location. This instruction indicates to the processor that the data is non-temporal, and is unlikely to be used again soon. The processor treats the store as a write-combining memory write, which minimizes cache pollution. The exact method by which cache pollution is minimized depends on the hardware implementation of the instruction. For further information, see “Memory Optimization” in volume 1.

Mnemonic	Opcode	Description
MOVNTQ *mem64*, mm*x*	0F E7 */r*	Stores a 64-bit MMX™ register value into a 64-bit memory location, minimizing cache pollution.

MOVNTQ is weakly-ordered with respect to other instructions that operate on memory. Software should use an SFENCE instruction to force strong memory ordering of MOVNTQ with respect to other stores.

MOVNTQ implicitly uses weakly-ordered, write-combining buffering for the data, as described in “Buffering and Combining Memory Writes” in volume 2. For data that is shared by multiple processors, this instruction should be used together with a fence instruction in order to ensure data coherency (refer to “Cache and TLB Management” in volume 2).

Related Instructions

MOVNTDQ, MOVNTI, MOVNTPD, MOVNTPS

Exceptions

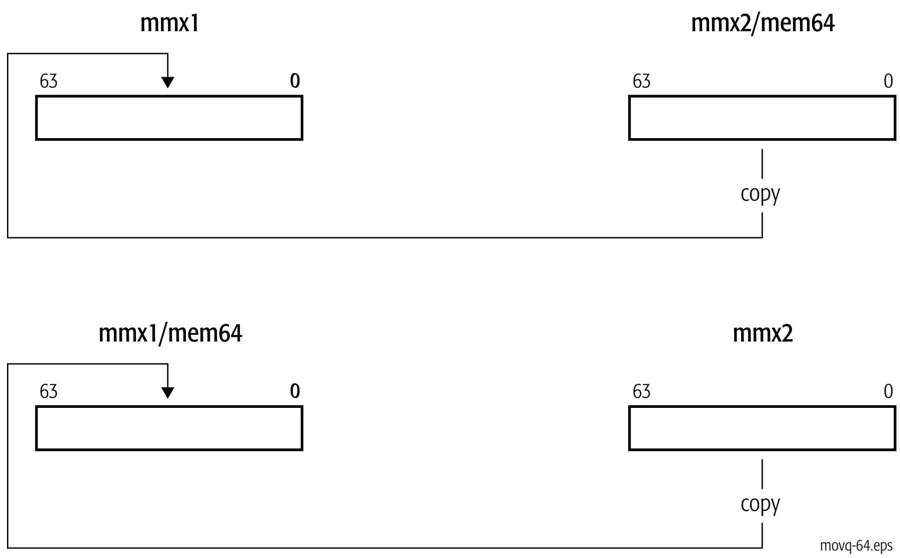
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The SSE instructions are not supported, as indicated by bit 25 in CPUID extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking is enabled.

MOVQ Move Quadword

The MOVQ instruction moves a 64-bit value:

- from an MMX register or 64-bit memory location to another MMX register, or
- from an MMX register to another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
MOVQ *mmx1*, *mmx2/mem64*	0F 6F /*r*	Moves 64-bit value from an MMX™ register or memory location to an MMX™ register.
MOVQ *mmx1/mem64*, *mmx2*	0F 7F /*r*	Moves 64-bit value from an MMX™ register to an MMX™ register or memory location.



Related Instructions

MOVD, MOVDQA, MOVDQU, MOVDQ2Q, MOVQ2DQ

rFLAGS Affected

None

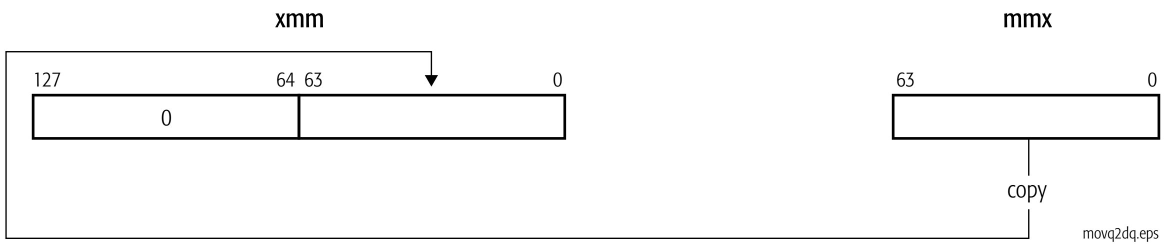
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The MMX instructions are not supported, as indicated by bit 23 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeds the stack segment limit or is non-canonical.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking is enabled.

MOVQ2DQ Move Quadword to Quadword

The MOVQ2DQ instruction moves a 64-bit value from an MMX register to the low-order 64 bits of an XMM register, with zero-extension to 128 bits.

Mnemonic	Opcode	Description
MOVQ2DQ *xmm*, *mmx*	F3 0F D6	Moves 64-bit value from an MMX™ register to an XMM register.



Execution of this instruction causes all fields in the x87 tag word to be set according to their corresponding data, the top-of-stack-pointer bit (TOP) in the x87 status word to be cleared to 0, and any pending x87 exceptions are handled before this instruction is executed. For details, see “Actions Taken on Executing 64-Bit Media Instructions” in volume 1.

Related Instructions

MOVD, MOVDQA, MOVDQU, MOVDQ2Q, MOVQ

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

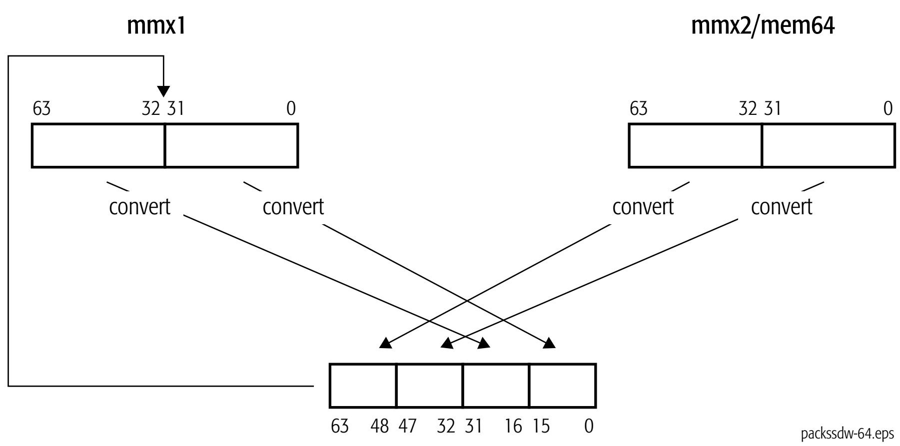
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 was cleared to 0. The SSE2 instructions are not supported, as indicated by bit 26 in CPUID extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.

PACKSSDW Pack with Saturation Signed Doubleword to Word

The PACKSSDW instruction converts each 32-bit signed integer in the first and second source operands to a 16-bit signed integer and packs the converted values into words in the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Converted values from the first source operand are packed into the low-order words of the destination, and the converted values from the second source operand are packed into the high-order words of the destination.

Mnemonic	Opcode	Description
PACKSSDW *mmx1*, *mmx2/mem64*	0F 6B /r	Packs 32-bit signed integers in an MMX™ register and another MMX™ register or 64-bit memory location into 16-bit signed integers in an MMX™ register.



For each packed value in the destination, if the value is larger than the largest signed 16-bit integer, it is saturated to 7FFFh, and if the value is smaller than the smallest signed 16-bit integer, it is saturated to 8000h.

Related Instructions

PACKSSWB, PACKUSWB

rFLAGS Affected

None

Exceptions

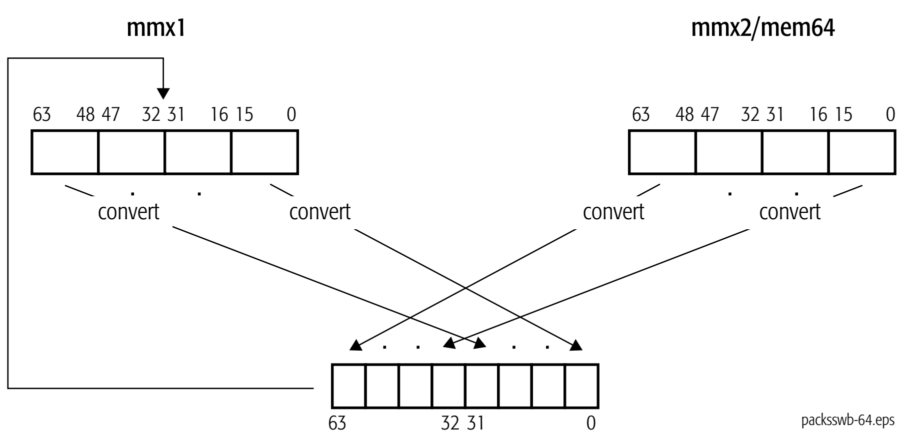
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The MMX instructions are not supported, as indicated by bit 23 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking is enabled.

PACKSSWB Pack with Saturation Signed Word to Byte

The PACKSSWB instruction converts each 16-bit signed integer in the first and second source operands to an 8-bit signed integer and packs the converted values into bytes in the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Converted values from the first source operand are packed into the low-order bytes of the destination, and the converted values from the second source operand are packed into the high-order bytes of the destination.

Mnemonic	Opcode	Description
PACKSSWB *mmx1*, *mmx2/mem64*	0F 63 */r*	Packs 16-bit signed integers in an MMX™ register and another MMX™ register or 64-bit memory location into 8-bit signed integers in an MMX™ register.



For each packed value in the destination, if the value is larger than the largest signed 8-bit integer, it is saturated to 7Fh, and if the value is smaller than the smallest signed 8-bit integer, it is saturated to 80h.

Related Instructions

PACKSSDW, PACKUSWB

rFLAGS Affected

None

Exceptions

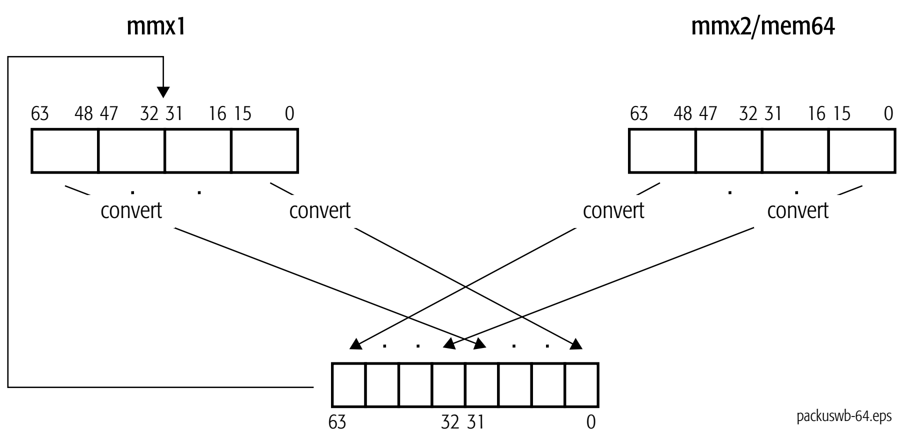
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The MMX instructions are not supported, as indicated by bit in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking is enabled.

PACKUSWB Pack with Saturation Signed Word to Unsigned Byte

The PACKUSWB instruction converts each 16-bit signed integer in the first and second source operands to an 8-bit unsigned integer and packs the converted values into bytes in the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Converted values from the first source operand are packed into the low-order bytes of the destination, and the converted values from the second source operand are packed into the high-order bytes of the destination.

Mnemonic	Opcode	Description
PACKUSWB *mmx1*, *mmx2/mem64*	0F 67 /*r*	Packs 16-bit signed integers in an MMX™ register and another MMX™ register or 64-bit memory location into 8-bit unsigned integers in an MMX™ register.



For each packed value in the destination, if the value is larger than the largest unsigned 8-bit integer, it is saturated to FFh, and if the value is smaller than the smallest unsigned 8-bit integer, it is saturated to 00h.

Related Instructions

PACKSSDW, PACKSSWB

rFLAGS Affected

None

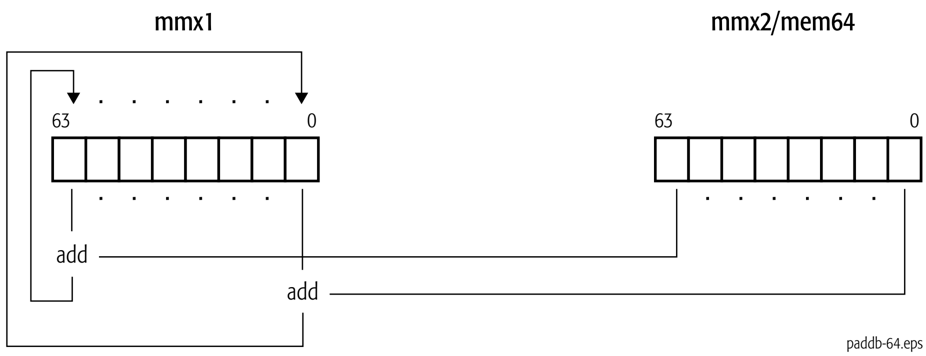
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The MMX instructions are not supported, as indicated by bit 23 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking is enabled.

PADDB Packed Add Bytes

The PADDB instruction adds each packed 8-bit integer value in the first source operand to the corresponding packed 8-bit integer in the second source operand and writes the integer result of each addition in the corresponding byte of the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PADDB *mmx1*, *mmx2/mem64*	0F FC /*r*	Adds packed byte integer values in an MMX™ register and another MMX™ register or 64-bit memory location and writes the result in the destination MMX™ register.



This instruction operates on both signed and unsigned integers. If the result overflows, the carry is ignored (neither the overflow nor carry bit in rFLAGS is set), and only the low-order 8 bits of each result are written in the destination.

Related Instructions

PADDD, PADDQ, PADDSB, PADDSW, PADDUSB, PADDUSW, PADDW

rFLAGS Affected

None

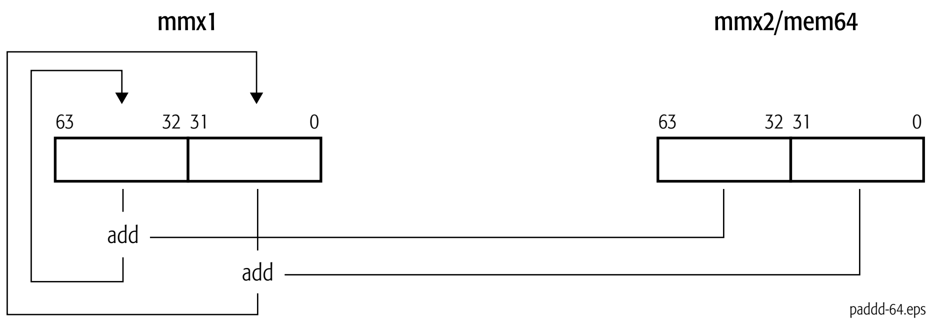
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The MMX instructions are not supported, as indicated by bit 23 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking is enabled.

PADDD Packed Add Doublewords

The PADDD instruction adds each packed 32-bit integer value in the first source operand to the corresponding packed 32-bit integer in the second source operand and writes the integer result of each addition in the corresponding doubleword of the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PADDD *mmx1*, *mmx2/mem64*	0F FE /*r*	Adds packed 32-bit integer values in an MMX™ register and another MMX™ register or 64-bit memory location and writes the result in the destination MMX™ register.



This instruction operates on both signed and unsigned integers. If the result overflows, the carry is ignored (neither the overflow nor carry bit in rFLAGS is set), and only the low-order 32 bits of each result are written in the destination.

Related Instructions

PADDB, PADDQ, PADDSB, PADDSW, PADDUSB, PADDUSW, PADDW

rFLAGS Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The MMX instructions are not supported, as indicated by bit 23 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking is enabled.

PADDQ Packed Add Quadwords

The PADDQ instruction adds each packed 64-bit integer value in the first source operand to the corresponding packed 64-bit integer in the second source operand and writes the integer result of each addition in the corresponding quadword of the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PADDQ *mmx1*, *mmx2/mem64*	0F D4 /r	Adds 64-bit integer value in an MMX™ register and another MMX™ register or 64-bit memory location and writes the result in the destination MMX™ register.

This instruction operates on both signed and unsigned integers. If the result overflows, the carry is ignored (neither the overflow nor carry bit in rFLAGS is set), and only the low-order 64 bits of each result are written in the destination.

Related Instructions

PADDB, PADDD, PADDSB, PADDSW, PADDUSB, PADDUSW, PADDW

rFLAGS Affected

None

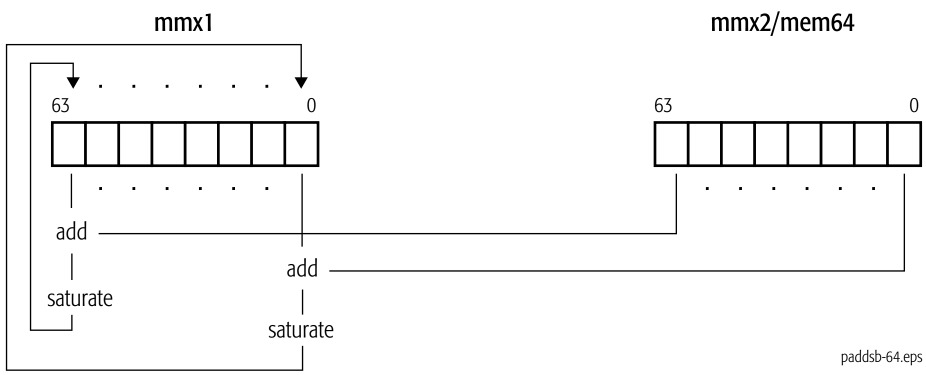
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The MMX instructions are not supported, as indicated by bit 23 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking is enabled.

PADDSB Packed Add Signed with Saturation Bytes

The PADDSB instruction adds each packed 8-bit signed integer value in the first source operand to the corresponding packed 8-bit signed integer in the second source operand and writes the signed integer result of each addition in the corresponding byte of the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PADDSB *mmx1*, *mmx2/mem64*	0F EC /*r*	Adds packed byte signed integer values in an MMX™ register and another MMX™ register or 64-bit memory location and writes the result in the destination MMX™ register.



For each packed value in the destination, if the value is larger than the largest representable signed 8-bit integer, it is saturated to 7Fh, and if the value is smaller than the smallest signed 8-bit integer, it is saturated to 80h.

Related Instructions

PADDB, PADDD, PADDQ, PADDSW, PADDUSB, PADDUSW, PADDW

rFLAGS Affected

None

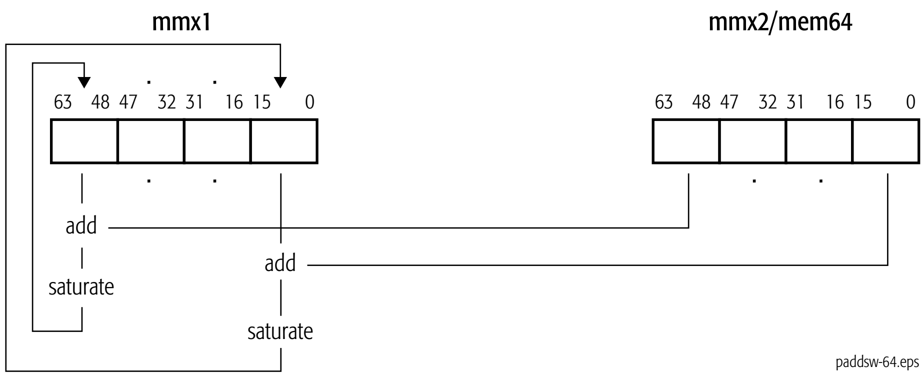
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The MMX instructions are not supported, as indicated by bit 23 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking is enabled.

PADDSW Packed Add Signed with Saturation Words

The PADDSW instruction adds each packed 16-bit signed integer value in the first source operand to the corresponding packed 16-bit signed integer in the second source operand and writes the signed integer result of each addition in the corresponding word of the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PADDSW *mmx1*, *mmx2/mem64*	0F ED */r*	Adds packed 16-bit signed integer values in an MMX™ register and another MMX™ register or 64-bit memory location and writes the result in the destination MMX™ register.



For each packed value in the destination, if the value is larger than the largest representable signed 16-bit integer, it is saturated to 7FFFh, and if the value is smaller than the smallest signed 16-bit integer, it is saturated to 8000h.

Related Instructions

PADDB, PADDD, PADDQ, PADDSB, PADDUSB, PADDUSW, PADDW

rFLAGS Affected

None

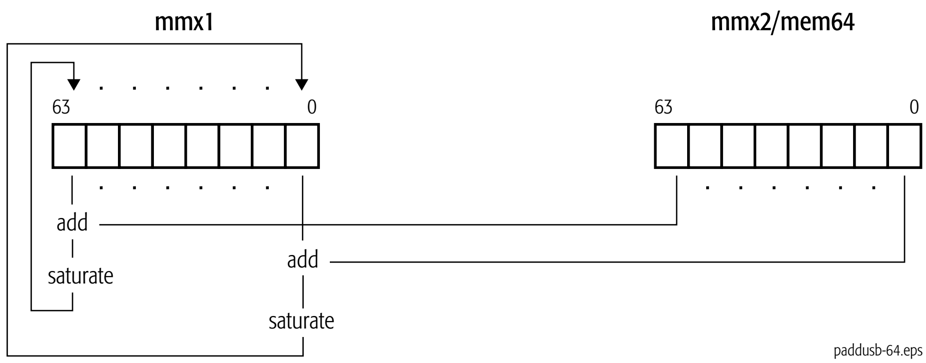
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The MMX instructions are not supported, as indicated by bit 23 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking is enabled.

PADDUSB Packed Add Unsigned with Saturation Bytes

The PADDUSB instruction adds each packed 8-bit unsigned integer value in the first source operand to the corresponding packed 8-bit unsigned integer in the second source operand and writes the unsigned integer result of each addition in the corresponding byte of the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PADDUSB *mmx1*, *mmx2/mem64*	0F DC /*r*	Adds packed byte unsigned integer values in an MMX™ register and another MMX™ register or 64-bit memory location and writes the result in the destination MMX™ register.



For each packed value in the destination, if the value is larger than the largest unsigned 8-bit integer, it is saturated to FFh, and if the value is smaller than the smallest unsigned 8-bit integer, it is saturated to 00h.

Related Instructions

PADDB, PADDD, PADDQ, PADDSB, PADDSW, PADDUSW, PADDW

rFLAGS Affected

None

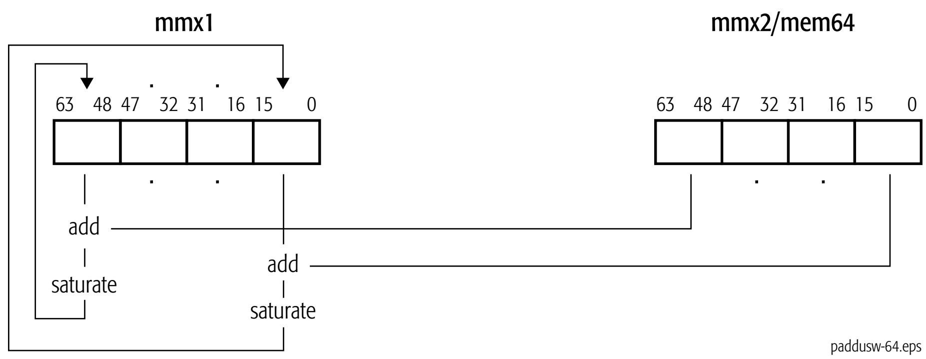
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The MMX instructions are not supported, as indicated by bit 23 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PADDUSW Packed Add Unsigned with Saturation Words

The PADDUSW instruction adds each packed 16-bit unsigned integer value in the first source operand to the corresponding packed 16-bit unsigned integer in the second source operand and writes the unsigned integer result of each addition in the corresponding word of the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PADDUSW *mmx1*, *mmx2/mem64*	0F DD /r	Adds packed 16-bit unsigned integer values in an MMX™ register and another MMX™ register or 64-bit memory location and writes result in the destination MMX™ register.



For each packed value in the destination, if the value is larger than the largest unsigned 16-bit integer, it is saturated to FFFFh, and if the value is smaller than the smallest unsigned 16-bit integer, it is saturated to 0000h.

Related Instructions

PADDB, PADDD, PADDQ, PADDSB, PADDSW, PADDUSB, PADDW

rFLAGS Affected

None

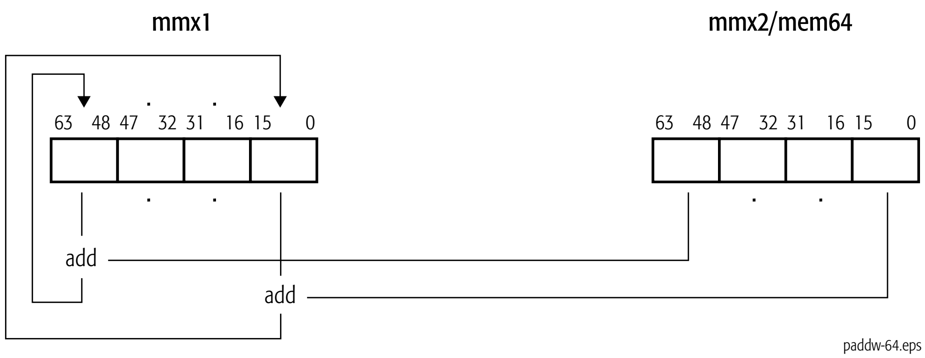
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The MMX instructions are not supported, as indicated by bit 23 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PADDW Packed Add Words

The PADDW instruction adds each packed 16-bit integer value in the first source operand to the corresponding packed 16-bit integer in the second source operand and writes the integer result of each addition in the corresponding word of the destination (second source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PADDW *mmx1*, *mmx2/mem64*	0F FD /r	Adds packed 16-bit integer values in an MMX™ register and another MMX™ register or 64-bit memory location and writes the result in the destination MMX™ register.



This instruction operates on both signed and unsigned integers. If the result overflows, the carry is ignored (neither the overflow nor carry bit in rFLAGS is set), and only the low-order 16 bits of the result are written in the destination.

Related Instructions

PADDB, PADDD, PADDQ, PADDSB, PADDSW, PADDUSB, PADDUSW

rFLAGS Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The MMX instructions are not supported, as indicated by bit 23 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PAND Packed Logical Bitwise AND

The PAND instruction performs a bitwise logical AND of the values in the first and second source operands and writes the result in the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PAND *mmx1*, *mmx2/mem64*	0F DB /*r*	Performs bitwise logical AND of values in an MMX™ register and in another MMX™ register or 64-bit memory location and writes the result in the destination MMX™ register.

Related Instructions

PANDN, POR, PXOR

rFLAGS Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The MMX instructions are not supported, as indicated by bit 23 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PANDN Packed Logical Bitwise AND NOT

The PANDN instruction performs a bitwise logical AND of the value in the second source operand and the one's complement of the value in the first source operand and writes the result in the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PANDN *mmx1*, *mmx2/mem64*	0F DF /r	Performs bitwise logical AND NOT of values in an MMX™ register and in another MMX™ register or 64-bit memory location and writes the result in the destination MMX™ register.

Related Instructions

PAND, POR, PXOR

rFLAGS Affected

None

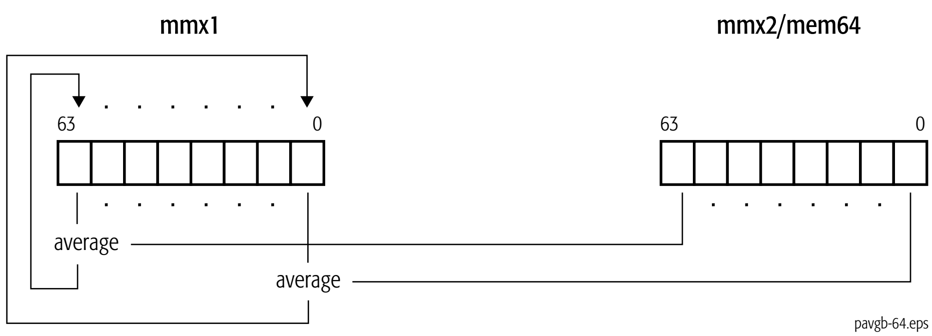
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The MMX instructions are not supported, as indicated by bit 23 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PAVGB Packed Average Unsigned Bytes

The PAVGB instruction computes the rounded average of each packed unsigned 8-bit integer value in the first source operand and the corresponding packed 8-bit unsigned integer in the second source operand and writes each average in the corresponding byte of the destination (first source). The average is computed by adding each pair of operands, adding 1 to the 9-bit temporary sum, and then right-shifting the temporary sum by one bit position. The destination and source operands are an MMX register and another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PAVGB *mmx1*, *mmx2/mem64*	0F E0 /*r*	Averages packed 8-bit unsigned integer values in an MMX™ register and another MMX™ register or 64-bit memory location and writes the result in the destination MMX™ register.



Related Instructions

PAVGW

rFLAGS Affected

None

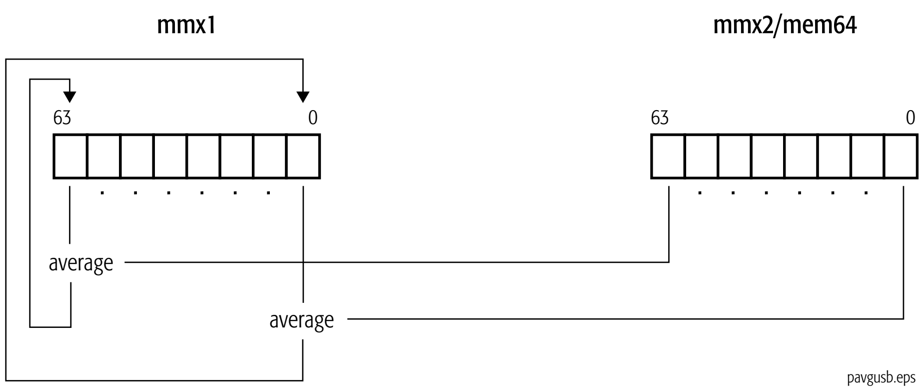
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The MMX instructions are not supported, as indicated by bit 23 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PAVGUSB Packed Average Unsigned Bytes

The PAVGUSB instruction computes the rounded-up average of each packed unsigned 8-bit integer value in the first source operand and the corresponding packed 8-bit unsigned integer in the second source operand and writes each average in the corresponding byte of the destination (first source). The average is computed by adding each pair of operands, adding 1 to the 9-bit temporary sum, and then right-shifting the temporary sum by one bit position. The first source/destination operand is an MMX register. The second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PAVGUSB *mmx1*, *mmx2/mem64*	0F 0F BF	Averages packed 8-bit unsigned integer values in an MMX™ register and another MMX™ register or 64-bit memory location and writes the result in the destination MMX™ register.



The PAVGUSB instruction performs a function identical to the 64-bit version of the PAVGB instruction, although the two instructions have different opcodes. PAVGUSB is a 3DNow! instruction. It is useful for pixel averaging in MPEG-2 motion compensation and video scaling operations.

Related Instructions

None

rFLAGS Affected

None

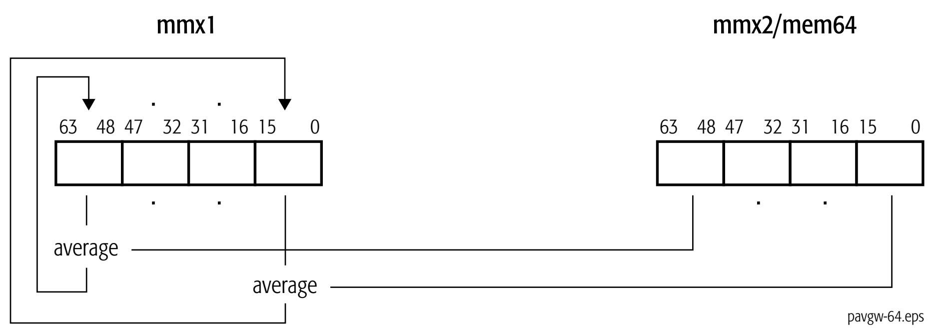
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The AMD 3DNow!™ instructions are not supported, as indicated by bit 31 in CPUID extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PAVGW Packed Average Unsigned Words

The PAVGW instruction computes the rounded average of each packed unsigned 16-bit integer value in the first source operand and the corresponding packed 16-bit unsigned integer in the second source operand and writes each average in the corresponding word of the destination (first source). The average is computed by adding each pair of operands, adding 1 to the 17-bit temporary sum, and then right-shifting the temporary sum by one bit position. The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PAVGW *mmx1*, *mmx2/mem64*	0F E3 */r*	Averages packed 16-bit unsigned integer values in an MMX™ register and another MMX™ register or 64-bit memory location and writes the result in the destination MMX™ register.



Related Instructions

PAVGB

rFLAGS Affected

None

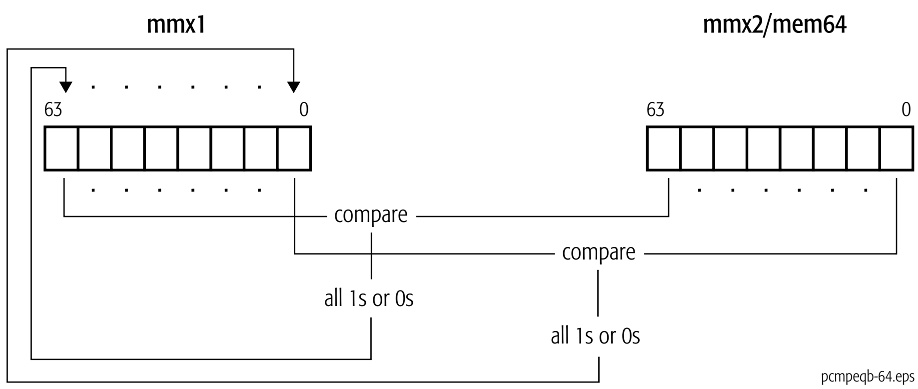
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The SSE instructions are not supported, as indicated by bit 25 in CPUID extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PCMPEQB Packed Compare Equal Bytes

The PCMPEQB instruction compares corresponding packed bytes in the first and second source operands and writes the result of each compare in the corresponding byte of the destination (first source). For each pair of bytes, if the values are equal, the result is all 1s. If the values are not equal, the result is all 0s. The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PCMPEQB *mmx1*, *mmx2/mem64*	0F 74 */r*	Compares packed bytes in an MMX™ register and an MMX™ register or 64-bit memory location.



Related Instructions

PCMPEQD, PCMPEQW, PCMPGTB, PCMPGTD, PCMPGTW

rFLAGS Affected

None

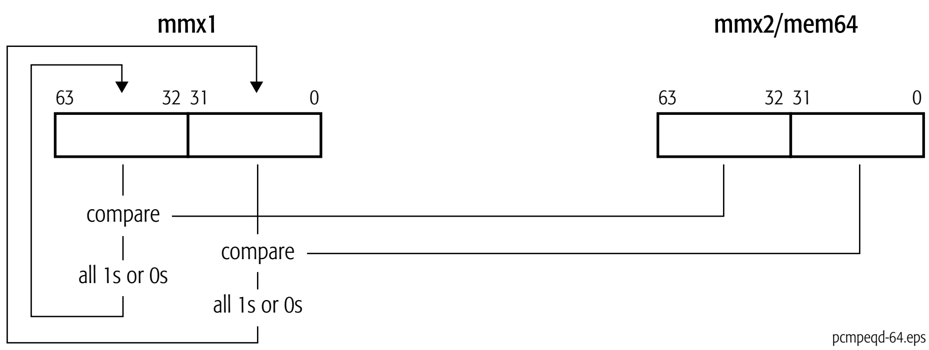
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The MMX instructions are not supported, as indicated by bit 23 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PCMPEQD Packed Compare Equal Doublewords

The PCMPEQD instruction compares corresponding packed 32-bit values in the first and second source operands and writes the result of each compare in the corresponding 32 bits of the destination (first source). For each pair of doublewords, if the values are equal, the result is all 1s. If the values are not equal, the result is all 0s. The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PCMPEQD *mmx1*, *mmx2/mem64*	0F 76 /r	Compares packed doublewords in an MMX™ register and an MMX™ register or 64-bit memory location.



Related Instructions

PCMPEQB, PCMPEQW, PCMPGTB, PCMPGTD, PCMPGTW

rFLAGS Affected

None

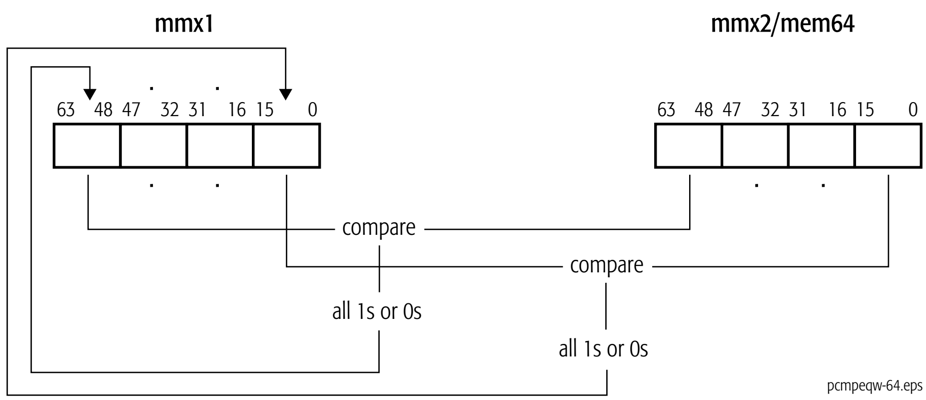
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The MMX instructions are not supported, as indicated by bit 23 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PCMPEQW Packed Compare Equal Words

The PCMPEQW instruction compares corresponding packed 16-bit values in the first and second source operands and writes the result of each compare in the corresponding 16 bits of the destination (first source). For each pair of words, if the values are equal, the result is all 1s. If the values are not equal, the result is all 0s. The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PCMPEQW *mmx1*, *mmx2/mem64*	0F 75 /r	Compares packed 16-bit values in an MMX™ register and an MMX™ register or 64-bit memory location.



Related Instructions

PCMPEQB, PCMPEQD, PCMPGTB, PCMPGTD, PCMPGTW

rFLAGS Affected

None

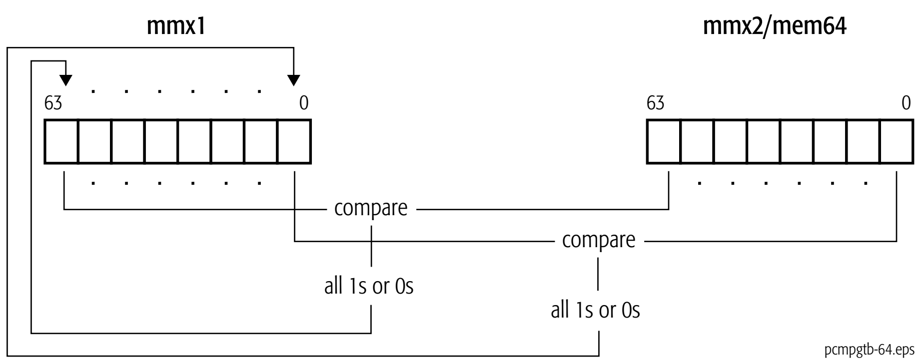
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The MMX instructions are not supported, as indicated by bit 23 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PCMPGTB Packed Compare Greater Than Signed Bytes

The PCMPGTB instruction compares corresponding packed signed bytes in the first and second source operands and writes the result of each compare in the corresponding byte of the destination (first source). For each pair of bytes, if the value in the first source operand is greater than the value in the second source operand, the result is all 1s. If the value in the first source operand is less than or equal to the value in the second source operand, the result is all 0s. The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PCMPGTB *mmx1*, *mmx2/mem64*	0F 64 /*r*	Compares packed signed bytes in an MMX™ register and an MMX™ register or 64-bit memory location.



Related Instructions

PCMPEQB, PCMPEQD, PCMPEQW, PCMPGTD, PCMPGTW

rFLAGS Affected

None

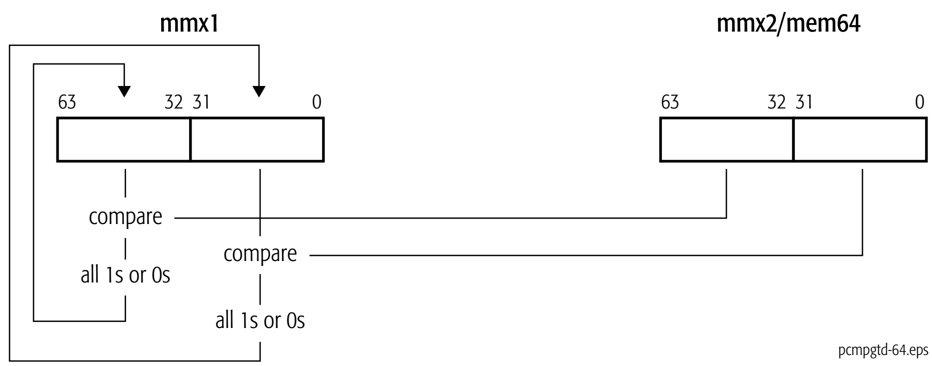
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The MMX instructions are not supported, as indicated by bit 23 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PCMPGTD Packed Compare Greater Than Signed Doublewords

The PCMPGTD instruction compares corresponding packed signed 32-bit values in the first and second source operands and writes the result of each compare in the corresponding 32 bits of the destination (first source). For each pair of doublewords, if the value in the first source operand is greater than the value in the second source operand, the result is all 1s. If the value in the first source operand is less than or equal to the value in the second source operand, the result is all 0s. The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PCMPGTD *mmx1*, *mmx2/mem64*	0F 66 */r*	Compares packed signed 32-bit values in an MMX™ register and an MMX™ register or 64-bit memory location.



Related Instructions

PCMPEQB, PCMPEQD, PCMPEQW, PCMPGTB, PCMPGTW

rFLAGS Affected

None

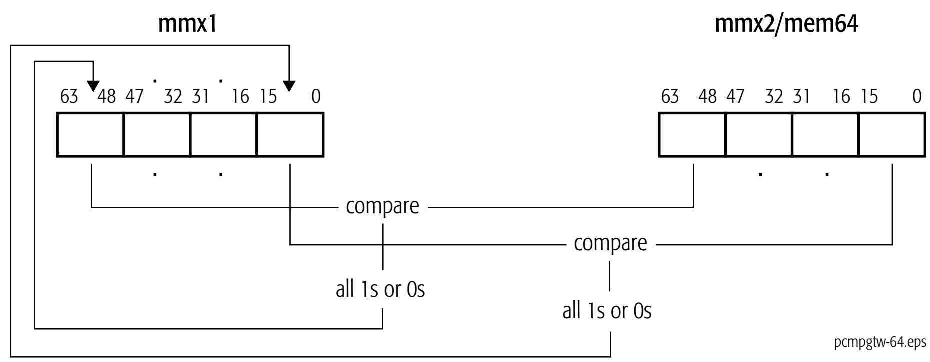
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The MMX instructions are not supported, as indicated by bit 23 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PCMPGTW Packed Compare Greater Than Signed Words

The PCMPGTW instruction compares corresponding packed signed 16-bit values in the first and second source operands and writes the result of each compare in the corresponding 16 bits of the destination (first source). For each pair of words, if the value in the first source operand is greater than the value in the second source operand, the result is all 1s. If the value in the first source operand is less than or equal to the value in the second source operand, the result is all 0s. The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PCMPGTW *mmx1*, *mmx2/mem64*	0F 65 /*r*	Compares packed signed 16-bit values in an MMX™ register and an MMX™ register or 64-bit memory location.



Related Instructions

PCMPEQB, PCMPEQD, PCMPEQW, PCMPGTB, PCMPGTD

rFLAGS Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The MMX instructions are not supported, as indicated by bit 23 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PEXTRW Extract Packed Word

The PEXTRW instruction extracts a 16-bit value from an MMX register, as selected by the immediate byte operand (as shown in Table 1-1) and writes it to the low-order word of a 32-bit or 64-bit general-purpose register, with zero-extension to 32 or 64 bits.

Mnemonic	Opcode	Description
PEXTRW *reg32/64*, *mmx*, *imm8*	0F C5 */r ib*	Extracts a 16-bit value from an MMX™ register and writes it to low-order 16 bits of a general-purpose register.

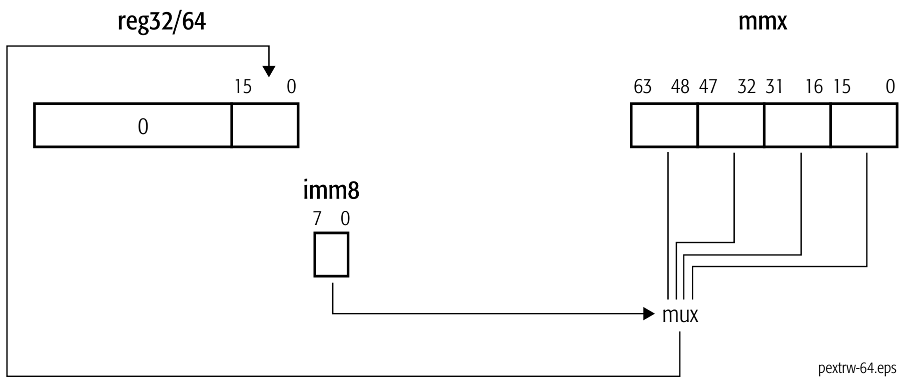


Table 1-1. Immediate-Byte Operand Encoding for 64-Bit PEXTRW

Immediate-Byte Bit Field	Value of Bit Field	Source Bits Extracted
1–0	0	15–0
	1	31–16
	2	47–32
	3	63–48

Related Instructions

PINSRW

rFLAGS Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The SSE instructions are not supported, as indicated by bit 25 in CPUID extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PF2ID Packed Floating-Point to Integer Doubleword Converson

The PF2ID instruction converts two packed single-precision floating-point values in an MMX register or a 64-bit memory location to two packed 32-bit signed integer values and writes the converted values in another MMX register. If the result of the conversion is an inexact value, the value is truncated (rounded toward zero). The numeric range for source and destination operands is shown in Table 1-2.

Mnemonic	Opcode	Description
PF2ID *mmx1*, *mmx2/mem64*	0F 0F 1D	Converts packed single-precision floating-point values in an MMX™ register or memory location to a doubleword integer value in the destination MMX™ register.

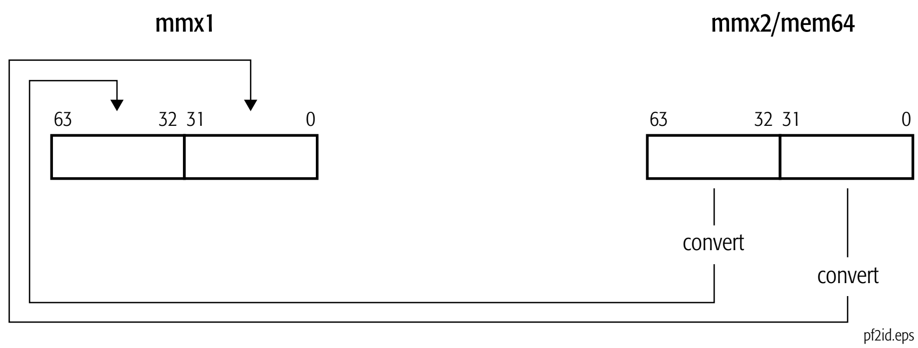


Table 1-2. Numeric Range for PF2ID Results

Source 2	Source 1 and Destination
0	0
Normal, abs(Source 1) < 1	0
Normal, -2^{31} < Source 1 <= –1	Round to zero (Source 1)
Normal, 1 <= Source 1 < 2^{31}	Round to zero (Source 1)
Normal, Source 1 >= 2^{31}	7FFF_FFFFh
Normal, Source 1 <= 2^{31}	8000_0000h
Unsupported	Undefined

Related Instructions

PF2IW, PI2FD, PI2FW

rFLAGS Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The AMD 3DNow!™ instructions are not supported, as indicated by bit 31 in CPUID extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PF2IW Packed Floating-Point to Integer Word Conversion

The PF2IW instruction converts two packed single-precision floating-point values in an MMX register or a 64-bit memory location to two packed 16-bit signed integer values, sign-extended to 32 bits, and writes the converted values in another MMX register. If the result of the conversion is an inexact value, the value is truncated (rounded toward zero). The numeric range for source and destination operands is shown in Table 1-3. Arguments outside the range representable by signed 16-bit integers are saturated to the largest and smallest 16-bit integer, depending on their sign.

Mnemonic	Opcode	Description
PF2IW *mmx1*, *mmx2/mem64*	0F 0F 1C	Converts packed single-precision floating-point values in an MMX™ register or memory location to word integer values in the destination MMX™ register.

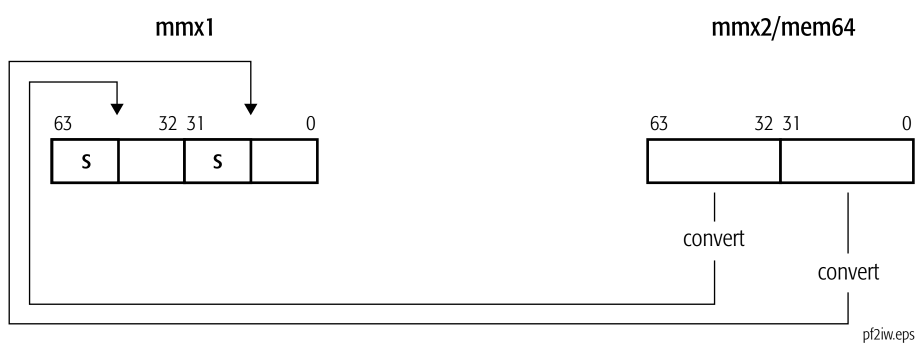


Table 1-3. Numeric Range for PF2IW Results

Source 2	Source 1 and Destination
0	0
Normal, abs(Source 1) < 1	0
Normal, -2^{15} < Source 1 <= –1	Round to zero (Source 1)
Normal, 1 <= Source 1 < 2^{15}	Round to zero (Source 1)

Table 1-3. Numeric Range for PF2IW Results

Source 2	Source 1 and Destination
Normal, Source 1 >= 2^{15}	7FFF_FFFFh
Normal, Source 1 <= 2^{15}	8000_0000h
Unsupported	Undefined

Related Instructions

PF2ID, PI2FD, PI2FW

rFLAGS Affected

None

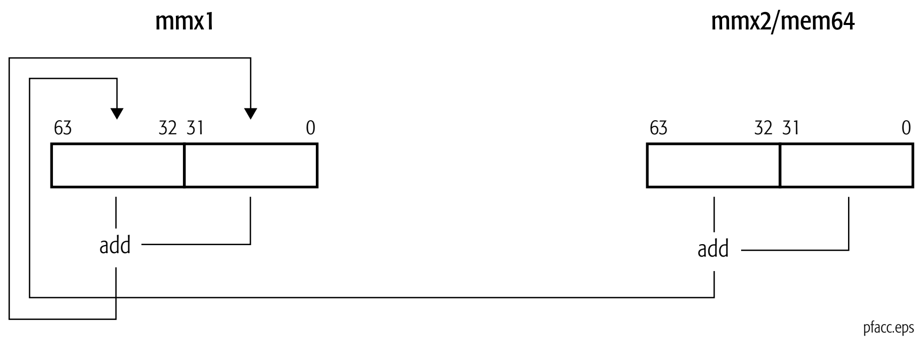
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The AMD Extensions to 3DNow!™ are not supported, as indicated by bit 30 in CPUID extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PFACC Packed Floating-Point Accumulate

The PFACC instruction adds the two single-precision floating-point values in the first source operand and adds the two single-precision values in the second source operand and writes the two results to the low-order and high-order doubleword, respectively, of the destination (first source). The first source/destination operand is an MMX register. The second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PFACC *mmx1*, *mmx2/mem64*	0F 0F AE	Accumulates packed single-precision floating-point values in an MMX™ register or 64-bit memory location and another MMX™ register and writes each result in the destination MMX™ register.



The numeric range for operands is shown in Table 1-4 on page 96.

Table 1-4. Numeric Range for PFACC Results

Source Operand		High Operand[2]		
		0	Normal	Unsupported
Low Operand[1]	0	+/– 0[3]	High Operand	High Operand
	Normal	Low Operand	Normal, +/– 0[4]	Undefined
	Unsupported[5]	Low Operand	Undefined	Undefined

Note:
1. *Least-significant floating-point value in first or second source operand.*
2. *Most-significant floating-point value in first or second source operand.*
3. *The sign of the result is the logical AND of the signs of the low and high operands*
4. *If the absolute value of the infinitely precise result is less than 2^{-126} (but not zero), the result is a zero with the sign of the operand (low or high) that is larger in magnitude. If the infinitely precise result is exactly zero, the result is zero with the sign of the low operand. If the absolute value of the infinitely precise result is greater than or equal to 2^{128}, the result is the largest normal number with the sign of the low operand.*
5. *"Unsupported" means that the exponent is all ones (1s).*

Related Instructions

PFADD, PFNACC, PFPNACC

rFLAGS Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The AMD 3DNow!™ instructions are not supported, as indicated bit 31 in CPUID extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PFADD Packed Floating-Point Add

The PFADD instruction adds each packed single-precision floating-point value in the first source operand to the corresponding packed single-precision floating-point value in the second operand and writes the result of each addition in the corresponding doubleword of the destination (first source). The first source/destination operand is an MMX register. The second source operand is another MMX register or 64-bit memory location. The numeric range for operands is shown in Table 1-5 on page 99.

Mnemonic	Opcode	Description
PFADD *mmx1*, *mmx2/mem64*	0F 0F 9E	Adds two packed single-precision floating-point values in an MMX™ register or 64-bit memory location and another MMX™ register and writes each result in the destination MMX™ register.

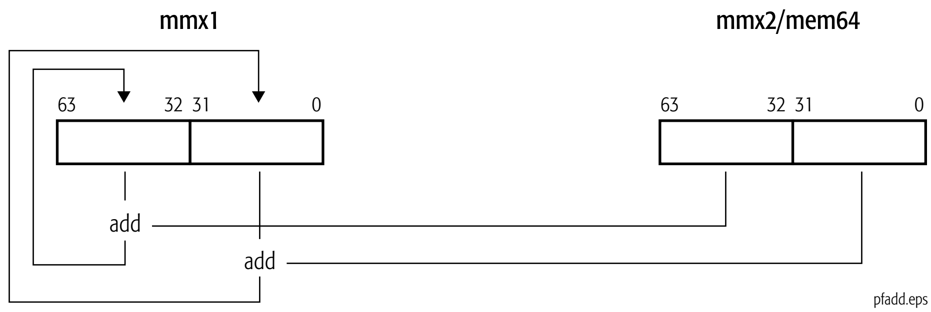


Table 1-5. Numeric Range for the PFADD Results

Source Operand		Most-Significant Doubleword		
		0	Normal	Unsupported
Source 1 and Destination	0	+/– 0[1]	Source 2	Source 2
	Normal	Source 1	Normal, +/– 0[2]	Undefined
	Unsupported[3]	Source 1	Undefined	Undefined

Note:

1. *The sign of the result is the logical AND of the signs of the source operands*
2. *If the absolute value of the infinitely precise result is less than 2^{-126} (but not zero), the result is a zero with the sign of the source operand that is larger in magnitude. If the infinitely precise result is exactly zero, the result is zero with the sign of source 1. If the absolute value of the infinitely precise result is greater than or equal to 2^{128}, the result is the largest normal number with the sign of source 1.*
3. *"Unsupported" means that the exponent is all ones (1s).*

Related Instructions

PFACC, PFNACC, PFPNACC

rFLAGS Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The AMD 3DNow!™ instructions are not supported, as indicated by bit 31 in CPUID extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PFCMPEQ Packed Floating-Point Compare Equal

The PFCMPEQ instruction compares each of the two packed single-precision floating-point values in the first source operand with the corresponding packed single-precision floating-point value in the second source operand and writes the result of each comparison in the corresponding doubleword of the destination (first source). For each pair of floating-point values, if the values are equal, the result is all 1s. If the values are not equal, the result is all 0s. The first source/destination operand is an MMX register. The second source operand is another MMX register or 64-bit memory location. The numeric range for operands is shown in Table 1-6 on page 101.

Mnemonic	Opcode	Description
PFCMPEQ *mmx1*, *mmx2/mem64*	0F 0F B0	Compares two pairs of packed single-precision floating-point values in an MMX™ register and an MMX™ register or 64-bit memory location.

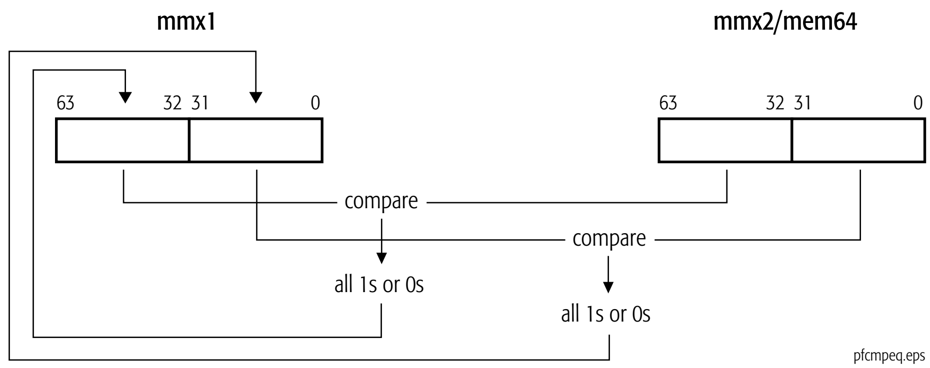


Table 1-6. Numeric Range for the PFCMPEQ Instruction

Operand	Value	Source 2		
		0	Normal	Unsupported
Source 1 and Destination	0	FFFF_FFFFh[1]	0000_0000h	0000_0000h
	Normal	0000_0000h	0000_0000h or FFFF_FFFFh[2]	0000_0000h
	Unsupported[3]	0000_0000h	0000_0000h	Undefined

Note:

1. *Positive zero is equal to negative zero.*
2. *The result is FFFF_FFFFh if source 1 and source 2 have identical signs, exponents, and mantissas. Otherwise, the result is 0000_0000h.*
3. *"Unsupported" means that the exponent is all ones (1s).*

Related Instructions

PFCMPGE, PFCMPGT

rFLAGS Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The AMD 3DNow!™ instructions are not supported, as indicated by bit 31 in CPUID extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PFCMPGE Packed Floating-Point Compare Greater or Equal

The PFCMPGE instruction compares each of the two packed single-precision floating-point values in the first source operand with the corresponding packed single-precision floating-point value in the second source operand and writes the result of each comparison in the corresponding doubleword of the destination (first source). For each pair of floating-point values, if the value in the first source operand is greater than or equal to the value in the second source operand, the result is all 1s. If the value in the first source operand is less than the value in the second source operand, the result is all 0s. The first source/destination operand is an MMX register. The second source operand is another MMX register or 64-bit memory location. The numeric range for operands is shown in Table 1-7 on page 103.

Mnemonic	Opcode	Description
PFCMPGE *mmx1*, *mmx2/mem64*	0F 0F 90	Compares two pairs of packed single-precision floating-point values in an MMX™ register and an MMX™ register or 64-bit memory location.

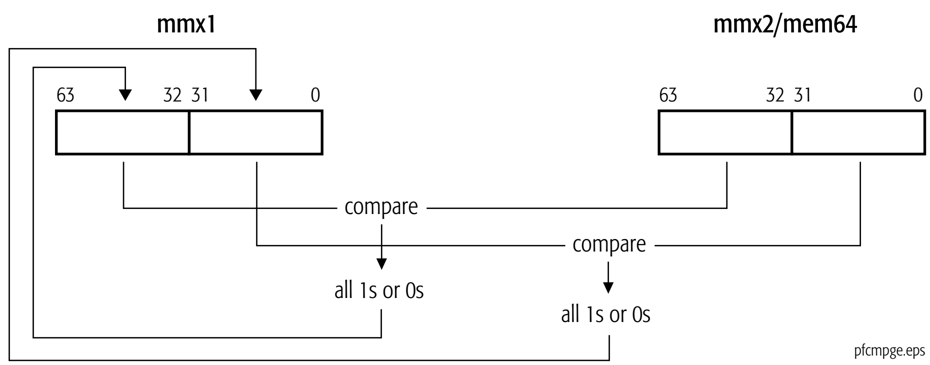


Table 1-7. Numeric Range for the PFCMPGE Instruction

Operand	Value	Source 2		
		0	Normal	Unsupported
Source 1 and Destination	0	FFFF_FFFFh[1]	0000_0000h, FFFF_FFFFh[2]	Undefined
	Normal	0000_0000h, FFFF_FFFFh[3]	0000_0000h, FFFF_FFFFh[4]	Undefined
	Unsupported[5]	Undefined	Undefined	Undefined

Note:

1. *Positive zero is equal to negative zero.*
2. *The result is FFFF_FFFFh, if source 2 is negative. Otherwise, the result is 0000_0000h.*
3. *The result is FFFF_FFFFh, if source 1 is positive. Otherwise, the result is 0000_0000h.*
4. *The result is FFFF_FFFFh, if source 1 is positive and source 2 is negative, or if they are both negative and source 1 is smaller than or equal in magnitude to source 2, or if source 1 and source 2 are both positive and source 1 is greater than or equal in magnitude to source 2. The result is 0000_0000h in all other cases.*
5. *"Unsupported" means that the exponent is all ones (1s).*

Related Instructions

PFCMPEQ, PFCMPGT

rFLAGS Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The AMD 3DNow!™ instructions are not supported, as indicated by bit 31 in CPUID extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PFCMPGT Packed Floating-Point Compare Greater Than

The PFCMPGT instruction compares each of the two packed single-precision floating-point values in the first source operand with the corresponding packed single-precision floating-point value in the second source operand and writes the result of each comparison in the corresponding doubleword of the destination (first source). For each pair of floating-point values, if the value in the first source operand is greater than the value in the second source operand, the result is all 1s. If the value in the first source operand is less than or equal to the value in the second source operand, the result is all 0s. The first source/destination operand is an MMX register. The second source operand is another MMX register or 64-bit memory location. The numeric range for operands is shown in Table 1-8 on page 106.

Mnemonic	Opcode	Description
PFCMPGT *mmx1*, *mmx2/mem64*	0F 0F A0	Compares two pairs of packed single-precision floating-point values in an MMX™ register and an MMX™ register or 64-bit memory location.

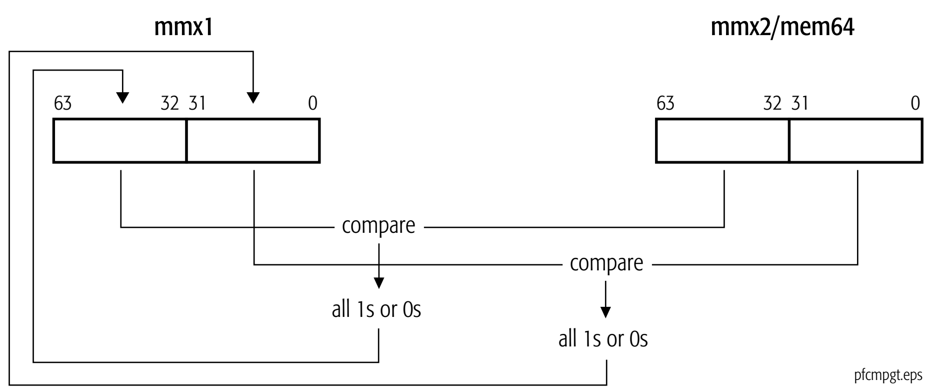


Table 1-8. Numeric Range for the PFCMPGT Instruction

Operand	Value	Source 2		
		0	Normal	Unsupported
Source 1 and Destination	0	0000_0000h	0000_0000h, FFFF_FFFFh[1]	Undefined
	Normal	0000_0000h, FFFF_FFFFh[2]	0000_0000h, FFFF_FFFFh[3]	Undefined
	Unsupported[4]	Undefined	Undefined	Undefined

Note:

1. *The result is FFFF_FFFFh, if source 2 is negative. Otherwise, the result is 0000_0000h.*
2. *The result is FFFF_FFFFh, if source 1 is positive. Otherwise, the result is 0000_0000h.*
3. *The result is FFFF_FFFFh, if source 1 is positive and source 2 is negative, or if they are both negative and source 1 is smaller in magnitude than source 2, or if source 1 and source 2 are positive and source 1 is greater in magnitude than source 2. The result is 0000_0000h in all other cases.*
4. *"Unsupported" means that the exponent is all ones (1s).*

Related Instructions

PFCMPEQ, PFCMPGE

rFLAGS Affected

None

Exceptions

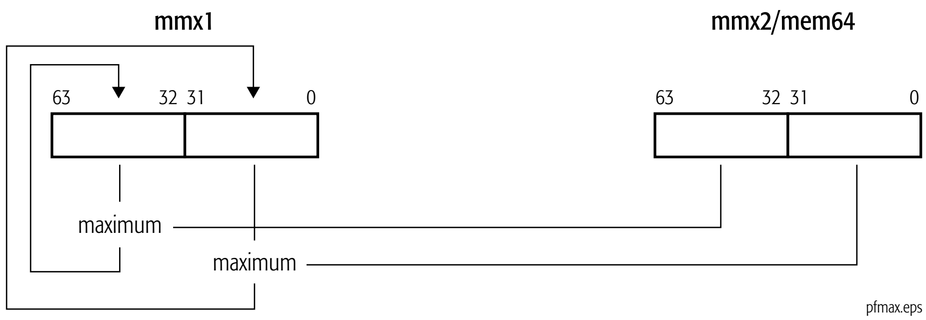
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The AMD 3DNow!™ instructions are not supported, as indicated by bit 31 in CPUID extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PFMAX — Packed Single-Precision Floating-Point Maximum

The PFMAX instruction compares each of the two packed single-precision floating-point values in the first source operand with the corresponding packed single-precision floating-point value in the second source operand and writes the maximum of the two values for each comparison in the corresponding doubleword of the destination (first source). The first source/destination operand is an MMX register. The second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PFMAX *mmx1*, *mmx2/mem64*	0F 0F A4	Compares two pairs of packed single-precision values in an MMX™ register and another MMX™ register or 64-bit memory location and writes the maximum value of each comparison in the destination MMX™ register.



Any operation with a zero and a negative number returns positive zero. An operation consisting of two zeros returns positive zero. If either source operand is an undefined value, the result is undefined. The numeric range for source and destination operands is shown in Table 1-9 on page 109.

Table 1-9. Numeric Range for the PFMAX Instruction

Operand	Value	Source 2		
		0	**Normal**	**Unsupported**
Source 1 and Destination	**0**	+0	Source 2, +0[1]	Undefined
	Normal	Source 1, +0[2]	Source 1/Source 2[3]	Undefined
	Unsupported[4]	Undefined	Undefined	Undefined

Note:

1. *The result is source 2, if source 2 is positive. Otherwise, the result is positive zero.*
2. *The result is source 1, if source 1 is positive. Otherwise, the result is positive zero.*
3. *The result is source 1, if source 1 is positive and source 2 is negative. The result is source 1, if both are positive and source 1 is greater in magnitude than source 2. The result is source 1, if both are negative and source 1 is lesser in magnitude than source 2. The result is source 2 in all other cases.*
4. *"Unsupported" means that the exponent is all ones (1s).*

Related Instructions

PFMIN

rFLAGS Affected

None

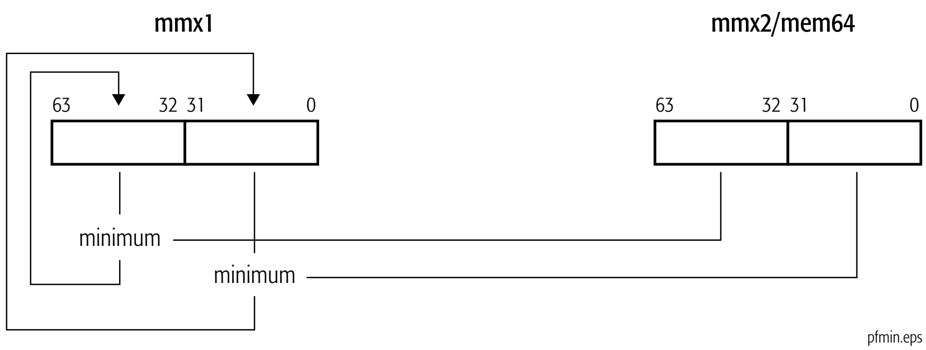
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The AMD 3DNow!™ instructions are not supported, as indicated by bit 31 in CPUID extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PFMIN Packed Single-Precision Floating-Point Minimum

The PFMIN instruction compares each of the two packed single-precision floating-point values in the first source operand with the corresponding packed single-precision floating-point value in the second source operand and writes the minimum of the two values for each comparison in the corresponding doubleword of the destination (first source). The first source/destination operand is an MMX register. The second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PFMIN *mmx1*, *mmx2/mem64*	0F 0F 94	Compares two pairs of packed single-precision values in an MMX™ register and another MMX™ register or 64-bit memory location and writes the minimum value of each comparison in the destination MMX™ register.



Any operation with a zero and a positive number returns positive zero. An operation consisting of two zeros returns positive zero. If either source operand is an undefined value, the result is undefined. The numeric range for source and destination operands is shown in Table 1-10 on page 111.

Table 1-10. Numeric Range for the PFMIN Instruction

Operand	Value	Source 2: 0	Source 2: Normal	Source 2: Unsupported
Source 1 and Destination	**0**	+0	Source 2, +0[1]	Undefined
	Normal	Source 1, +0[2]	Source 1/Source 2[3]	Undefined
	Unsupported[4]	Undefined	Undefined	Undefined

Note:

1. *The result is source 2, if source 2 is negative. Otherwise, the result is positive zero.*
2. *The result is source 1, if source 1 is negative. Otherwise, the result is positive zero.*
3. *The result is source 1, if source 1 is negative and source 2 is positive. The result is source 1, if both are negative and source 1 is greater in magnitude than source 2. The result is source 1, if both are positive and source 1 is lesser in magnitude than source 2. The result is source 2 in all other cases.*
4. *"Unsupported" means that the exponent is all ones (1s).*

Related Instructions

PFMAX

rFLAGS Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The AMD 3DNow!™ instructions are not supported, as indicated by bit 31 in CPUID extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PFMUL Packed Floating-Point Multiply

The PFMUL instruction multiplies each of the two packed single-precision floating-point values in the first source operand by the corresponding packed single-precision floating-point value in the second source operand and writes the result of each multiplication in the corresponding doubleword of the destination (first source). The numeric range for source and destination operands is shown in Table 1-11 on page 113. The first source/destination operand is an MMX register. The second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PFMUL *mmx1*, *mmx2/mem64*	0F 0F B4	Multiplies packed single-precision floating-point values in an MMX™ register and another XMM register or 64-bit memory location and writes the result in the destination MMX™ register.

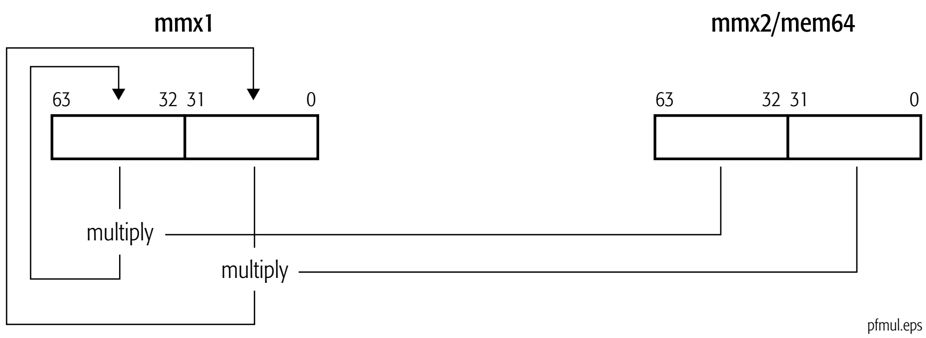


Table 1-11. Numeric Range for the PFMUL Instruction

Operand	Value	Source 2		
		0	**Normal**	**Unsupported**
Source 1 and Destination	**0**	+/– 0[1]	+/– 0[1]	+/– 0[1]
	Normal	+/– 0[1]	Normal, +/– 0[2]	Undefined
	Unsupported[3]	+/– 0[1]	Undefined	Undefined

Note:

1. *The sign of the result is the exclusive-OR of the signs of the source operands.*
2. *If the absolute value of the result is less then 2^{-126}, the result is zero with the sign being the exclusive-OR of the signs of the source operands. If the absolute value of the product is greater than or equal to 2^{128}, the result is the largest normal number with the sign being the exclusive-OR of the signs of the source operands.*
3. *"Unsupported" means that the exponent is all ones (1s).*

Related Instructions

None

rFLAGS Affected

None

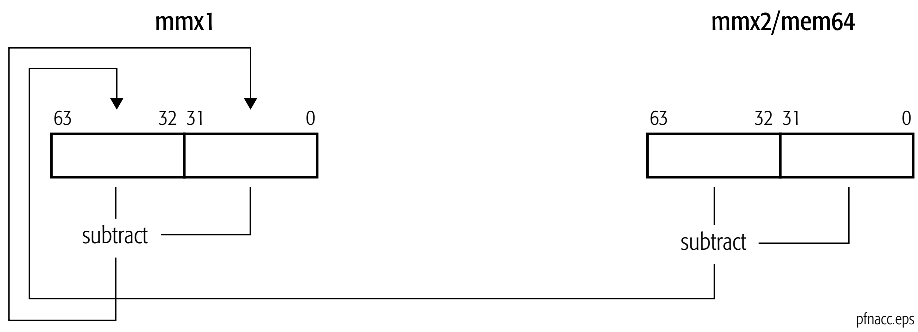
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The AMD 3DNow!™ instructions are not supported, as indicated by bit 31 in CPUID extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PFNACC Packed Floating-Point Negative Accumulate

The PFNACC instruction subtracts the first source operand's high-order single-precision floating-point value from its low-order single-precision floating-point value, subtracts the second source operand's high-order single-precision floating-point value from its low-order single-precision floating-point value, and writes each result to the low-order or high-order doubleword, respectively, of the destination (first source). The first source/destination operand is an MMX register. The second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PFNACC *mmx1*, *mmx2/mem64*	0F 0F 8A	Subtracts the packed single-precision floating-point values in an MMX™ register or 64-bit memory location and another MMX™ register and writes each value in the destination MMX™ register.



The numeric range for operands is shown in Table 1-12 on page 115.

Table 1-12. Numeric Range of PFNACC Results

Source Operand		High Operand[2]		
		0	**Normal**	**Unsupported**
Low Operand[1]	**0**	+/– 0^3	- High Operand	- High Operand
	Normal	Low Operand	Normal, +/– 0^4	Undefined
	Unsupported[5]	Low Operand	Undefined	Undefined

Note:

1. *Least-significant floating-point value in first or second source operand.*
2. *Most-significant floating-point value in first or second source operand.*
3. *The sign is the logical AND of the sign of the low operand and the inverse of the sign of the high operand.*
4. *If the absolute value of the infinitely precise result is less than 2^{-126} (but not zero), the result is a zero. If the low operand is larger in magnitude than the high operand, the sign of this zero is the same as the sign of the low operand, else it is the inverse of the sign of the high operand. If the infinitely precise result is exactly zero, the result is zero with the sign of the low operand. If the absolute value of the infinitely precise result is greater than or equal to 2^{128}, the result is the largest normal number with the sign of the low operand.*
5. *"Unsupported" means that the exponent is all ones (1s).*

Related Instructions

PFSUB, PFACC, PFPNACC

rFLAGS Affected

None

Exceptions

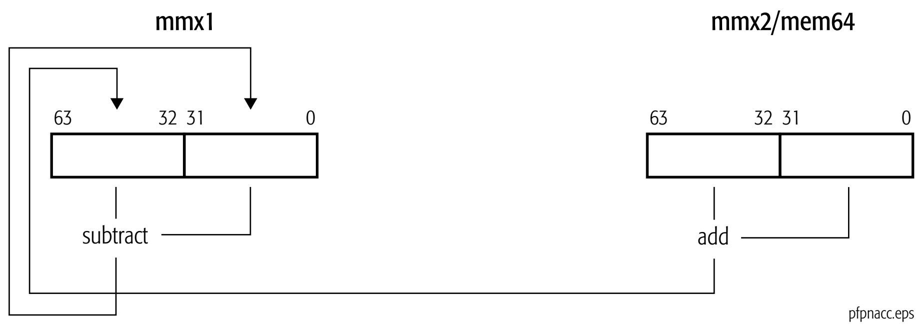
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The AMD Extensions to 3DNow!™ are not supported, as indicated by bit 30 in CPUID extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PFPNACC Packed Floating-Point Positive-Negative Accumulate

The PFPNACC instruction subtracts the first source operand's high-order single-precision floating-point value from its low-order single-precision floating-point value, adds the two single-precision values in the second source operand, and writes each result to the low-order or high-order doubleword, respectively, of the destination (first source). The first source/destination operand is an MMX register. The second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PFPNACC *mmx1*, *mmx2/mem64*	0F 0F 8E	Subtracts the packed single-precision floating-point values in an MMX™ register, adds the packed single-precision floating-point values in another MMX™ register or 64-bit memory location, and writes each value in the destination MMX™ register.



The numeric range for operands is shown in Table 1-13 (for the low result) and Table 1-14 (for the high result), both on page 118.

Table 1-13. Numeric Range of PFPNACC Result (Low Result)

Source Operand		High Operand[2] 0	High Operand[2] Normal	High Operand[2] Unsupported
Low Operand[1]	0	+/– 0^3	- High Operand	- High Operand
	Normal	Low Operand	Normal, +/– 0^4	Undefined
	Unsupported[5]	Low Operand	Undefined	Undefined

Note:
1. *Least-significant floating-point value in first or second source operand.*
2. *Most-significant floating-point value in first or second source operand.*
3. *The sign is the logical AND of the sign of the low operand and the inverse of the sign of the high operand.*
4. *If the absolute value of the infinitely precise result is less than 2^{-126} (but not zero), the result is a zero. If the low operand is larger in magnitude than the high operand, the sign of this zero is the same as the sign of the low operand, else it is the inverse of the sign of the high operand. If the infinitely precise result is exactly zero, the result is zero with the sign of the low operand. If the absolute value of the infinitely precise result is greater than or equal to 2^{128}, the result is the largest normal number with the sign of the low operand.*
5. *"Unsupported" means that the exponent is all ones (1s).*

Table 1-14. Numeric Range of PFPNACC Result (High Result)

Source Operand		High Operand[2] 0	High Operand[2] Normal	High Operand[2] Unsupported
Low Operand[1]	0	+/– 0^3	High Operand	High Operand
	Normal	Low Operand	Normal, +/– 0^4	Undefined
	Unsupported[5]	Low Operand	Undefined	Undefined

Note:
1. *Least-significant floating-point value in first or second source operand.*
2. *Most-significant floating-point value in first or second source operand.*
3. *The sign is the logical AND of the signs of the low and high operands.*
4. *If the absolute value of the infinitely precise result is less than 2^{-126} (but not zero), the result is zero with the sign of the operand (low or high) that is larger in magnitude. If the infinitely precise result is exactly zero, the result is zero with the sign of the low operand. If the absolute value of the infinitely precise result is greater than or equal to 2^{128}, the result is the largest normal number with the sign of the low operand.*
5. *"Unsupported" means that the exponent is all ones (1s).*

Related Instructions

PFADD, PFSUB, PFACC, PFNACC

rFLAGS Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The AMD Extensions to 3DNow!™ are not supported, as indicated by bit 30 in CPUID extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PFRCP Floating-Point Reciprocal Approximation

The PFRCP instruction computes the approximate reciprocal of the single-precision floating-point value in the low-order 32 bits of an MMX register or 64-bit memory location and writes the result in both doublewords of another MMX register. The result is accurate to 14 bits.

Mnemonic	Opcode	Description
PFRCP *mmx1*, *mmx2/mem64*	0F 0F 96	Computes approximate reciprocal of single-precision floating-point value in an MMX™ register or 64-bit memory location and writes the result in both doublewords of the destination MMX™ register.

The PFRCP result can forwarded to the Newton-Raphson iteration step 1 (PFRCPIT1) and Newton-Raphson iteration step 2 (PFRCPIT2) instructions to increase the accuracy of the reciprocal. The first stage of this refinement in accuracy (PFRCPIT1) requires that the input and output of the previously executed PFRCP instruction be used as input to the PFRCPIT1 instruction.

The estimate contains the correct round-to-nearest value for approximately 99% of all arguments. The remaining arguments differ from the correct round-to-nearest value for the reciprocal by 1 unit-in-the-last-place (ulp). For details, see the data sheet or other software-optimization documentation relating to particular hardware implementations.

PFRCP(x) returns 0 for x >= 2^{-126}. The numeric range for operands is shown in Table 1-15.

Table 1-15. Numeric Range for the PFRCP Result

Operand		Source 1 and Destination
Source 2	0	+/– Maximum Normal[1]
	Normal	Normal, +/– 0[2]
	Unsupported[3]	Undefined

Note:
1. *The result has the same sign as the source operand.*
2. *If the absolute value of the result is less then 2^{-126}, the result is zero with the sign being the sign of the source operand. Otherwise, the result is a normal with the sign being the same sign as the source operand.*
3. *"Unsupported" means that the exponent is all ones (1s).*

Examples

The general Newton-Raphson recurrence for the reciprocal 1/b is:

$$Z_{i+1} \leftarrow Z_i \bullet (2 - b \bullet Z_i)$$

The following code sequence shows the computation of a/b:

X_0 = PFRCP(b)
X_1 = PFRCPIT1(b, X_0)
X_2 = PFRCPIT2(X_1, X_0)
q = PFMUL(a, X_2)

The 24-bit final reciprocal value is X_2. The quotient is formed in the last step by multiplying the reciprocal by the dividend a.

Related Instructions

PFRCPIT1, PFRCPIT2

rFLAGS Affected

None

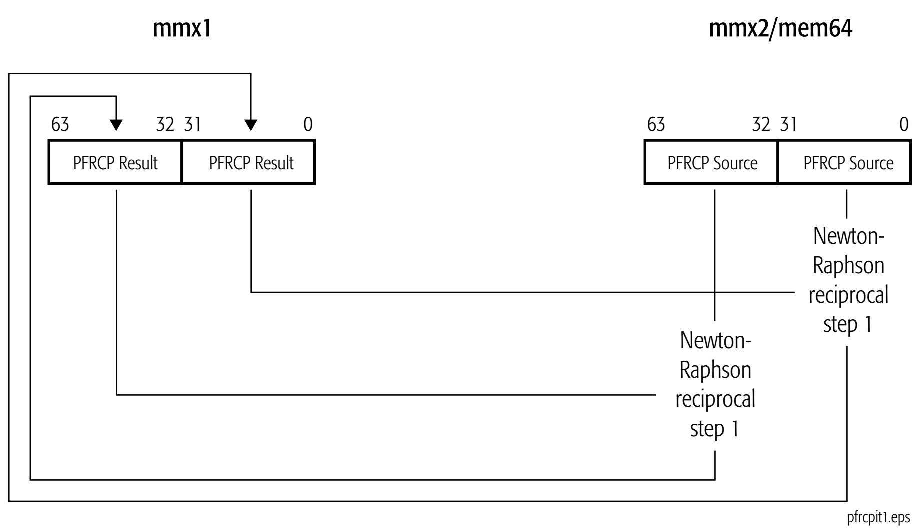
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The AMD 3DNow!™ instructions are not supported, as indicated by bit 31 in CPUID extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PFRCPIT1 Packed Floating-Point Reciprocal Iteration 1

The PFRCPIT1 instruction performs the first step in the Newton-Raphson iteration to refine the reciprocal approximation produced by the PFRCP instruction. The first source/destination operand is an MMX register containing the results of two previous PFRCP instructions, and the second source operand is another MMX register or 64-bit memory location containing the source operands from the same PFRCP instructions.

Mnemonic	Opcode	Description
PFRCPIT1 *mmx1*, *mmx2/mem64*	0F 0F A6	Refine approximate reciprocal of result from previous PFRCP instruction.



This instruction is only defined for those combinations of operands such that the first source operand (mmx1) is the approximate reciprocal of the second source operand (mmx2/mem64), and thus the range of the product, mmx1 * mmx2/mem64, is (0.5, 2). The initial approximation of an operand is accurate to about 12 bits, and the length of the operand itself is 24 bits, so the product of these two operands is greater than 24 bits. PFRCPIT1 applies the one's complement of the product and rounds the result to 32 bits. It then compresses the result to fit into 24 bits by removing the 8 redundant most-significant bits after the hidden integer bit.

The estimate contains the correct round-to-nearest value for approximately 99% of all arguments. The remaining arguments differ from the correct round-to-nearest value for the reciprocal by 1 unit-in-the-last-place (ulp). For details, see the data sheet or

other software-optimization documentation relating to particular hardware implementations.

The numeric range for operands is shown in Table 1-16.

Table 1-16. Numeric Range for the PFRCPIT1 Results

Source Operand		Source 2		
		0	Normal	Unsupported
Source 1 and Destination	0	+/– 0[1]	+/– 0[1]	+/– 0[1]
	Normal	+/– 0[1]	Normal[2]	Undefined
	Unsupported[3]	+/– 0[1]	Undefined	Undefined

Note:
1. *The sign of the result is the exclusive-OR of the signs of the source operands.*
2. *The sign is positive.*
3. *"Unsupported" means that the exponent is all ones (1s).*

Operation

```
mmx1[31:0] = Compress (2 - mmx1[31:0] * (mmx2/mem64[31:0]) - 2^31);
mmx1[63:32] = Compress (2 - mmx1[63:32] * (mmx2/mem64[63:32]) - 2^31);
```

where:

"Compress" means discard the 8 redundant most-significant bits after the hidden integer bit.

Examples

The general Newton-Raphson recurrence for the reciprocal 1/b is:

$Z_{i+1} \leftarrow Z_i \bullet (2 - b \bullet Z_i)$

The following code sequence computes a 24-bit approximation to a/b with one Newton-Raphson iteration:

```
X0 = PFRCP(b)
X1 = PFRCPIT1(b, X0)
X2 = PFRCPIT2(X1, X0)
q = PFMUL(a, X2)
```

a/b is formed in the last step by multiplying the reciprocal approximation by a.

Related Instructions

PFRCP, PFRCPIT2

rFLAGS Affected

None

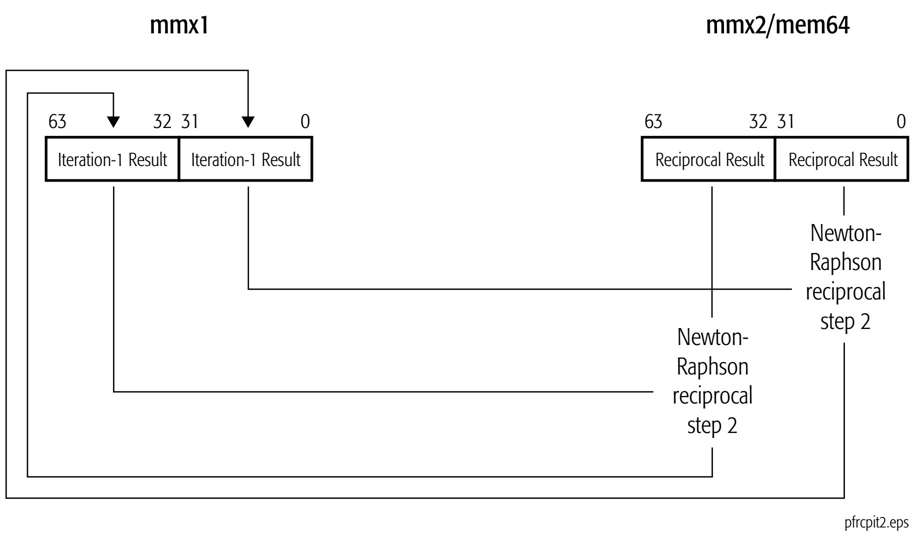
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The AMD 3DNow!™ instructions are not supported, as indicated by bit 31 in CPUID extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PFRCPIT2 Packed Floating-Point Reciprocal or Reciprocal Square Root Iteration 2

The PFRCPIT2 instruction performs the second and final step in the Newton-Raphson iteration to refine the reciprocal approximation produced by the PFRCP instruction or the reciprocal square-root approximation produced by the PFSQRT instruction. PFRCPIT2 takes two paired elements in each source operand. These paired elements are the results of a PFRCP and PFRCPIT1 instruction sequence or of a PFRSQRT and PFRSQIT1 instruction sequence. The first source/destination operand is an MMX register that contains the PFRCPIT1 or PFRSQIT1 results and the second source operand is another MMX register or 64-bit memory location that contains the PFRCP or PFRSQRT results.

Mnemonic	Opcode	Description
PFRCPIT2 *mmx1*, *mmx2/mem64*	0F 0F B6	Refines approximate reciprocal result from previous PFRCP and PFRCPIT1 instructions or from previous PFRSQRT and PFRSQIT1 instructions.



The PFRCPIT2 instruction expands the compressed PFRCPIT1 or PFRSQIT1 results from 24 to 32 bits and multiplies them by their respective source operands. An optimal correction factor is added to the product, which is then rounded to 24 bits.

The estimate contains the correct round-to-nearest value for approximately 99% of all arguments. The remaining arguments differ from the correct round-to-nearest value for the reciprocal by 1 unit-in-the-last-place (ulp). For details, see the data sheet or other software-optimization documentation relating to particular hardware implementations.

The numeric range for operands is shown in Table 1-17.

Table 1-17. Numeric Range for the PFRCPIT2 Results

Source Operand		Source 2		
		0	**Normal**	**Unsupported**
Source 1 and Destination	**0**	+/– 0[1]	+/– 0[1]	+/– 0[1]
	Normal	+/– 0[1]	Normal, +/– 0[2]	Undefined
	Unsupported[3]	+/– 0[1]	Undefined	Undefined

Note:
1. *The sign of the result is the exclusive-OR of the signs of the source operands.*
2. *If the absolute value of the result is less then 2^{-126}, the result is zero with the sign being the exclusive-OR of the signs of the source operands. If the absolute value of the product is greater than or equal to 2^{128}, the result is the largest normal number with the sign being exclusive-OR of the signs of the source operands.*
3. *“Unsupported” means that the exponent is all ones (1s).*

Operation

```
mmx1[31:0] = Expand(mmx1[31:0]) * mmx2/mem64[31:0];
mmx1[63:32] = Expand(mmx1[63:32]) * mmx2/mem64[63:32];
```

where:

“Expand” means convert a 24-bit significand to a 32-bit significand according to the following rule:

```
temp[31:0] = {1’b1, 8{mmx1[22]}, mmx1[22:0]};
```

Examples

The general Newton-Raphson recurrence for the reciprocal 1/b is:

$Z_{i+1} \leftarrow Z_i \bullet (2 - b \bullet Z_i)$

The following code sequence computes a 24-bit approximation to a/b with one Newton-Raphson iteration:

X_0 = PFRCP(b)
X_1 = PFRCPIT1(b, X_0)
X_2 = PFRCPIT2(X_1, X_0)

```
q = PFMUL(a, X2)
```

a/b is formed in the last step by multiplying the reciprocal approximation by a.

Related Instructions

PFRCP, PFRCPIT1, PFRSQRT, PFRSQIT1

rFLAGS Affected

None

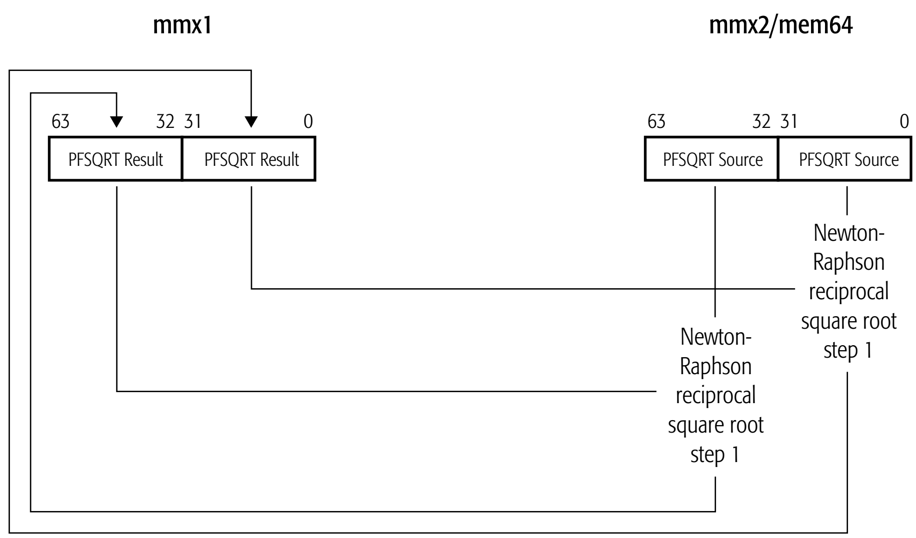
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The AMD 3DNow!™ instructions are not supported, as indicated by bit 31 in CPUID extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PFRSQIT1 Packed Floating-Point Reciprocal Square Root Iteration 1

The PFRSQIT1 instruction performs the first step in the Newton-Raphson iteration to refine the reciprocal square-root approximation produced by the PFSQRT instruction. The first source/destination operand is an MMX register containing the result from a previous PFRSQRT instruction, and the second source operand is another MMX register or 64-bit memory location containing the source operand from the same PFRSQRT instruction.

Mnemonic	Opcode	Description
PFRSQIT1 *mmx1*, *mmx2/mem64*	0F 0F A7	Refines reciprocal square root approximation of previous PFRSQRT instruction.



pfrsqit1.eps

This instruction is only defined for those combinations of operands such that the first source operand (mmx1) is the approximate reciprocal of the second source operand (mmx2/mem64), and thus the range of the product, mmx1 * mmx2/mem64, is (0.5, 2). The length of both operands is 24 bits, so the product of these two operands is greater than 24 bits. The product is normalized and then rounded to 32 bits. The one's complement of the result is applied, a 1 is added as the most-significant bit, and the result re-normalized. The result is then compressed to fit into 24 bits by removing 8

redundant most-significant bits after the hidden integer bit, and the exponent is reduced by 1 to account for the division by 2.

The numeric range for operands is shown in Table 1-18.

Table 1-18. Numeric Range for the PFRSQIT1 Result

Source Operand		Source 2: 0	Source 2: Normal	Source 2: Unsupported
Source 1 and Destination	0	+/– 0[1]	+/– 0[1]	+/– 0[1]
	Normal	+/– 0[1]	Normal[2]	Undefined
	Unsupported[3]	+/– 0[1]	Undefined	Undefined

Note:

1. *The sign of the result is the exclusive-OR of the signs of the source operands.*
2. *The sign is 0.*
3. *"Unsupported" means that the exponent is all ones (1s).*

Operation

```
mmx1[31:0] = Compress ((3 - mmx1[31:0] * (mmx2/mem64[31:0]) - 2^31)/2);
mmx1[63:32] = Compress ((3 - mmx1[63:32] * (mmx2/mem64[63:32]) - 2^31)/2);
```

where:

"Compress" means discard the 8 redundant most-significant bits after the hidden integer bit.

Examples

The following code sequence shows how the PFRSQRT and PFMUL instructions can be used to compute a = 1/sqrt (b):

```
X0 = PFRSQRT(b)
X1 = PFMUL(X0,X0)
X2 = PFRSQIT1(b,X1)
a = PFRCPIT2(X2,X0)
```

Related Instructions

PFRCPIT2, PFRSQRT

rFLAGS Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The AMD 3DNow!™ instructions are not supported, as indicated by bit 31 in CPUID extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PFRSQRT Packed Floating-Point Reciprocal Square Root Approximation

The PFRSQRT instruction computes the approximate reciprocal square root of the single-precision floating-point value in the low-order 32 bits of an MMX register or 64-bit memory location and writes the result in each doubleword of another MMX register. The source operand is single-precision with a 24-bit significand, and the result is accurate to 15 bits. Negative operands are treated as positive operands for purposes of reciprocal square-root computation, with the sign of the result the same as the sign of the source operand.

Mnemonic	Opcode	Description
PFRSQRT *mmx1*, *mmx2/mem64*	0F 0F 97	Computes approximate reciprocal square root of a packed single-precision floating-point value.

This instruction can be used together with the PFRSQIT1 and PFRCPIT2 instructions to increase accuracy. The first stage of this refinement in accuracy (PFRSQIT1) requires that the input and output of the previously executed PFRSQRT instruction be used as input to the PFRSQIT1 instruction.

The estimate contains the correct round-to-nearest value for approximately 99% of all arguments. The remaining arguments differ from the correct round-to-nearest value for the reciprocal by 1 unit-in-the-last-place (ulp). For details, see the data sheet or other software-optimization documentation relating to particular hardware implementations.

The numeric range for operands is shown in Table 1-19 on page 133.

Table 1-19. Numeric Range for the PFRCP Result

Operand		Source 1 and Destination
Source 2	0	+/– Maximum Normal[1]
	Normal	Normal[1]
	Unsupported[2]	Undefined[1]

Note:
1. *The result has the same sign as the source operand.*
2. *"Unsupported" means that the exponent is all ones (1s).*

Examples

The following code sequence shows how the PFRSQRT and PFMUL instructions can be used to compute a = 1/sqrt (b):

X_0 = PFRSQRT(b)
X_1 = PFMUL(X_0,X_0)
X_2 = PFRSQIT1(b,X_1)
a = PFRCPIT2(X_2,X_0)

Related Instructions

PFRCPIT2, PFRSQIT1

rFLAGS Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The AMD 3DNow!™ instructions are not supported, as indicated by bit 31 in CPUID extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PFSUB Packed Floating-Point Subtract

The PFSUB instruction subtracts each packed single-precision floating-point value in the second source operand from the corresponding packed single-precision floating-point value in the first source operand and writes the result of each subtraction in the corresponding doubleword of the destination (first source). The first source/destination operand is an MMX register. The second source operand is another MMX register or 64-bit memory location. The numeric range for operands is shown in Table 1-20 on page 136.

Mnemonic	Opcode	Description
PFSUB *mmx1*, *mmx2/mem64*	0F 0F 9A	Subtracts packed single-precision floating-point values in an MMX™ register or 64-bit memory location from packed single-precision floating-point values in another MMX™ register and writes the result in the destination MMX™ register.

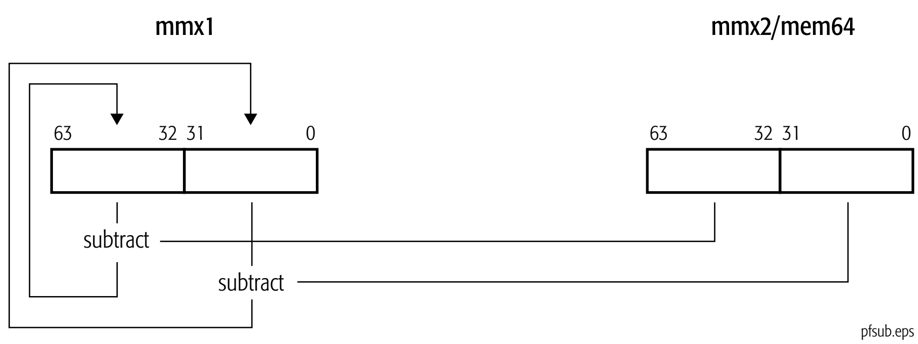


Table 1-20. Numeric Range for the PFSUB Results

Source Operand		Source 2: 0	Source 2: Normal	Source 2: Unsupported
Source 1 and Destination	0	+/– 0[1]	- Source 2	- Source 2
	Normal	Source 1	Normal, +/– 0[2]	Undefined
	Unsupported[3]	Source 1	Undefined	Undefined

Note:

1. *The sign of the result is the logical AND of the sign of source 1 and the inverse of the sign of source 2*
2. *If the absolute value of the infinitely precise result is less than 2^{-126} (but not zero), the result is a zero. If the source operand that is larger in magnitude is source 1, the sign of this zero is the same as the sign of source 1, else it is the inverse of the sign of source 2. If the infinitely precise result is exactly zero, the result is zero with the sign of source 1. If the absolute value of the infinitely precise result is greater than or equal to 2^{128}, the result is the largest normal number with the sign of source 1.*
3. *"Unsupported" means that the exponent is all ones (1s).*

Related Instructions

PFSUBR

rFLAGS Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The AMD 3DNow!™ instructions are not supported, as indicated by bit 31 in CPUID extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PFSUBR Packed Floating-Point Subtract Reverse

The PFSUBR instruction subtracts each packed single-precision floating-point value in the first source operand from the corresponding packed single-precision floating-point value in the second source operand and writes the result of each subtraction in the corresponding quadword of the destination (first source). The first source/destination operand is an MMX register. The second source operand is another MMX register or 64-bit memory location. The numeric range for operands is shown in Table 1-21 on page 139.

Mnemonic	Opcode	Description
PFSUBR *mmx1*, *mmx2/mem64*	0F 0F AA	Subtracts packed single-precision floating-point values in an MMX™ register from packed single-precision floating-point values in another MMX™ register or 64-bit memory location and writes the result in the destination MMX™ register.

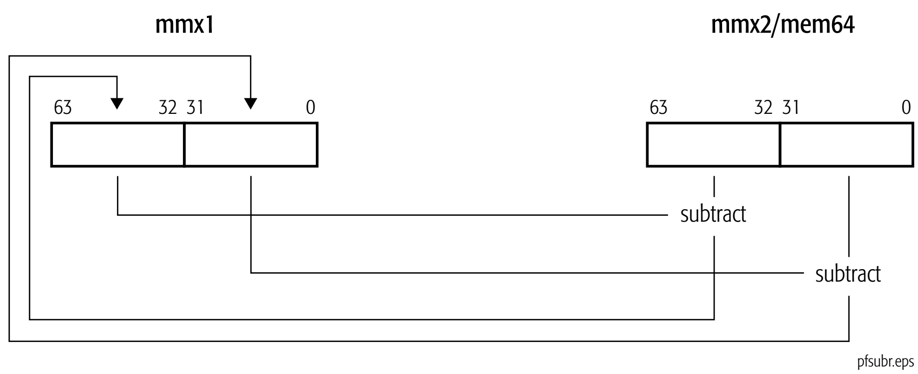


Table 1-21. Numeric Range for the PFSUBR Results

Source Operand		Source 2		
		0	**Normal**	**Unsupported**
Source 1 and Destination	**0**	+/– 0[1]	Source 2	Source 2
	Normal	- Source 1	Normal, +/– 0[2]	Undefined
	Unsupported[3]	- Source 1	Undefined	Undefined

Note:

1. *The sign is the logical AND of the sign of source 2 and the inverse of the sign of source 1.*
2. *If the absolute value of the infinitely precise result is less than 2^{-126} (but not zero), the result is a zero. If the source operand that is larger in magnitude is source 2, the sign of this zero is the same as the sign of source 2, else it is the inverse of the sign of source 1. If the infinitely precise result is exactly zero, the result is zero with the sign of source 2. If the absolute value of the infinitely precise result is greater than or equal to 2^{128}, the result is the largest normal number with the sign of source 2.*
3. *"Unsupported" means that the exponent is all ones (1s).*

Related Instructions

PFSUB

rFLAGS Affected

None

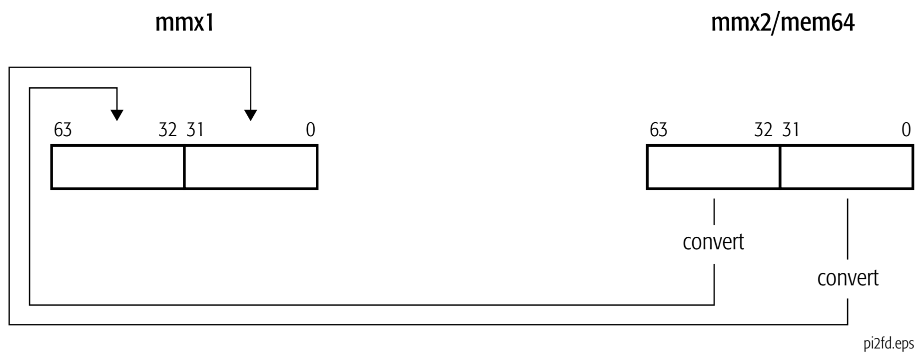
Exceptions0

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The AMD 3DNow!™ instructions are not supported, as indicated by bit 31 in CPUID extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PI2FD Packed Integer to Floating-Point Doubleword Conversion

The PI2FD instruction converts two packed 32-bit signed integer values in an MMX register or a 64-bit memory location to two packed single-precision floating-point values and writes the converted values in another MMX register. If the result of the conversion is an inexact value, the value is truncated (rounded toward zero).

Mnemonic	Opcode	Description
PI2FD *mmx1*, *mmx2/mem64*	0F 0F 0D	Converts packed doubleword integers in an MMX™ register or 64-bit memory location to single-precision floating-point values in the destination MMX™ register. Inexact results are truncated.



Related Instructions

PF2ID, PF2IW, PI2FW

rFLAGS Affected

None

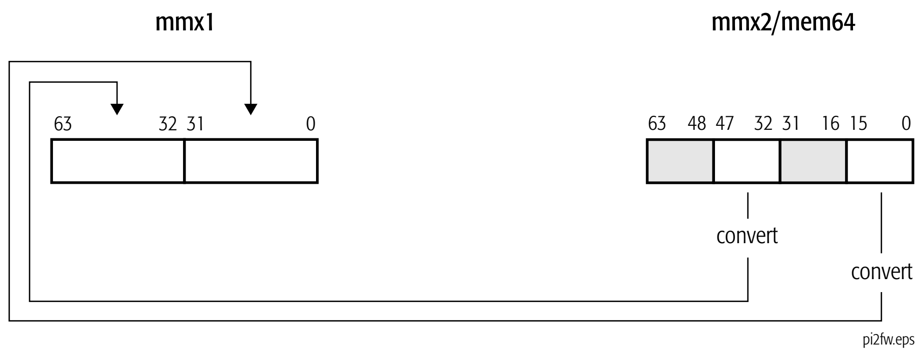
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The AMD 3DNow!™ instructions are not supported, as indicated by bit 31 in CPUID extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PI2FW Packed Integer to Floating-Point Word Conversion

The PI2FW instruction converts two packed 16-bit signed integer values in an MMX register or a 64-bit memory location to two packed single-precision floating-point values and writes the converted values in another MMX register.

Mnemonic	Opcode	Description
PI2FW *mmx1*, *mmx2/mem64*	0F 0F 0C	Converts packed 16-bit integers in an XMM register or 64-bit memory location to packed single-precision floating-point values in the destination MMX™ register.



Related Instructions

PF2ID, PF2IW, PI2FD

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The AMD Extensions to 3DNow!™ are not supported, as indicated by bit 30 in CPUID extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PINSRW Packed Insert Word

The PINSRW instruction inserts a 16-bit value from the low-order word of a 32-bit or 64-bit general purpose register or a 16-bit memory location into an MMX register. The location in the destination register is selected by the immediate byte operand, a shown in Table 1-22 on page 144. The other words in the destination register operand are not modified.

Mnemonic	Opcode	Description
PINSRW *mmx, reg32/64/mem16*, *imm8*	0F C4 */r ib*	Inserts a 16-bit value from a general-purpose register or memory location into an MMX™ register.

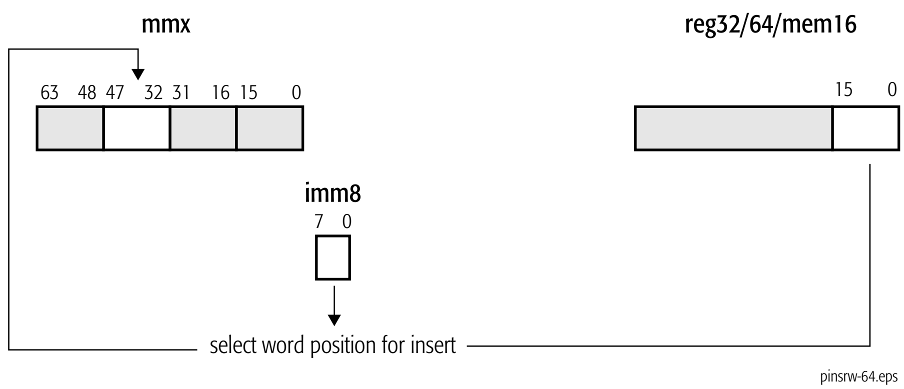


Table 1-22. Immediate-Byte Operand Encoding for 64-Bit PINSRW

Immediate-Byte Bit Field	Value of Bit Field	Destination Bits Filled
1–0	0	15–0
	1	31–16
	2	47–32
	3	63–48

Related Instructions

PEXTRW

rFLAGS Affected

None

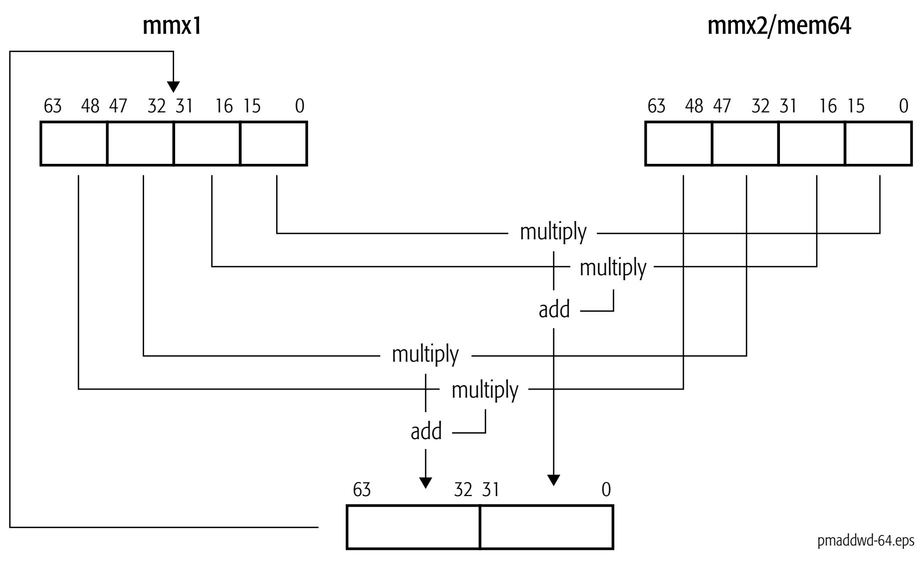
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The SSE instructions are not supported, as indicated by bit 25 in CPUID extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PMADDWD Packed Multiply Words and Add Doublewords

The PMADDWD instruction multiplies each packed 16-bit signed value in the first source operand by the corresponding packed 16-bit signed value in the second source operand, adds the adjacent intermediate 32-bit results of each multiplication (for example, the multiplication results for the adjacent bit fields 63–48 and 47–32, and 31–16 and 15–0), and writes the 32-bit result of each addition in the corresponding doubleword of the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PMADDWD *mmx1*, *mmx2/mem64*	0F F5 /*r*	Multiplies four packed 16-bit signed values in an MMX™ register and another MMX™ register or 64-bit memory location, adds intermediate results, and writes the result in the destination MMX™ register.



If all four of the 16-bit source operands used to produce a 32-bit multiply-add result have the value 8000h, the 32-bit result is 8000_0000h, which is incorrect.

Related Instructions

PMULHUW, PMULHW, PMULLW, PMULUDQ

rFLAGS Affected

None

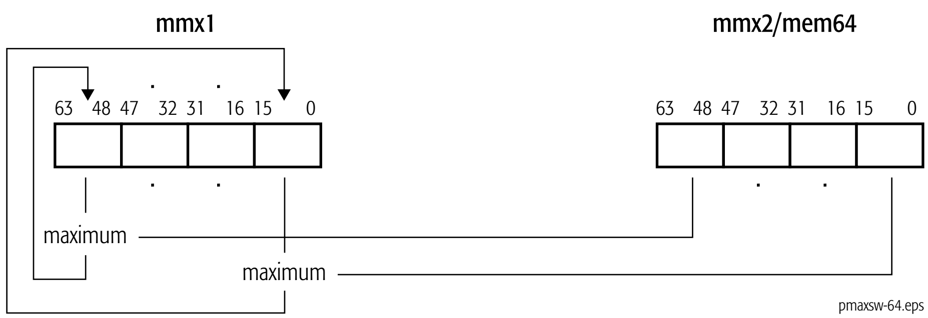
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The MMX instructions are not supported, as indicated by bit 23 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PMAXSW Packed Maximum Signed Words

The PMAXSW instruction compares each of the packed 16-bit signed integer values in the first source operand with the corresponding packed 16-bit signed integer value in the second source operand and writes the maximum of the two values for each comparison in the corresponding word of the destination (first source). The first source/destination and second source operands are an MMX register and an MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PMAXSW *mmx1*, *mmx2/mem64*	0F EE /*r*	Compares packed signed 16-bit integer values in an MMX™ register and another MMX™ register or 64-bit memory location and writes the maximum value of each compare in destination MMX™ register.



Related Instructions

PMAXUB, PMINSW, PMINUB

rFLAGS Affected

None

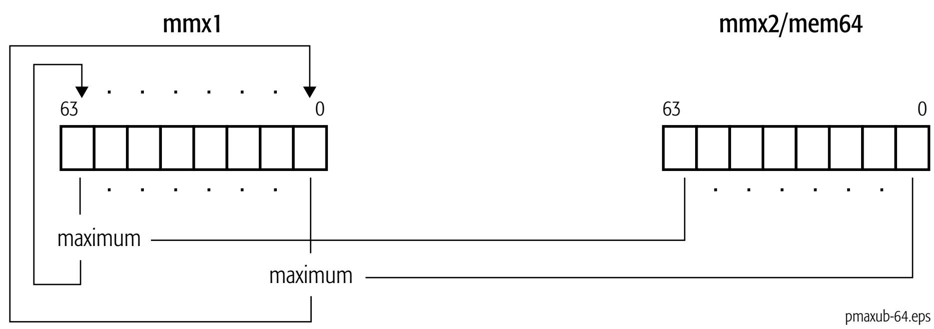
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The SSE instructions are not supported, as indicated by bit 25 in CPUID extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PMAXUB Packed Maximum Unsigned Bytes

The PMAXUB instruction compares each of the packed 8-bit unsigned integer values in the first source operand with the corresponding packed 8-bit unsigned integer value in the second source operand and writes the maximum of the two values for each comparison in the corresponding byte of the destination (first source). The first source/destination and second source operands are an MMX register and an MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PMAXUB *mmx1*, *mmx2/mem64*	0F DE /*r*	Compares packed unsigned 8-bit integer values in an MMX™ register and another MMX™ register or 64-bit memory location and writes the maximum value of each compare in the destination MMX™ register.



Related Instructions

PMAXSW, PMINSW, PMINUB

rFLAGS Affected

None

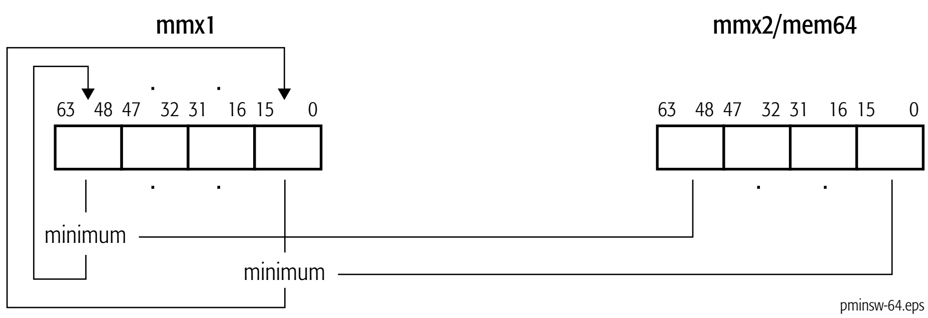
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The SSE instructions are not supported, as indicated by bit 25 in CPUID extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PMINSW Packed Minimum Signed Words

The PMINSW instruction compares each of the packed 16-bit signed integer values in the first source operand with the corresponding packed 16-bit signed integer value in the second source operand and writes the minimum of the two values for each comparison in the corresponding word of the destination (first source). The first source/destination and second source operands are an MMX register and an MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PMINSW *mmx1*, *mmx2/mem64*	0F EA /*r*	Compares packed signed 16-bit integer values in an MMX™ register and another MMX™ register or 64-bit memory location and writes the minimum value of each compare in the destination MMX™ register.



Related Instructions

PMAXSW, PMAXUB, PMINUB

rFLAGS Affected

None

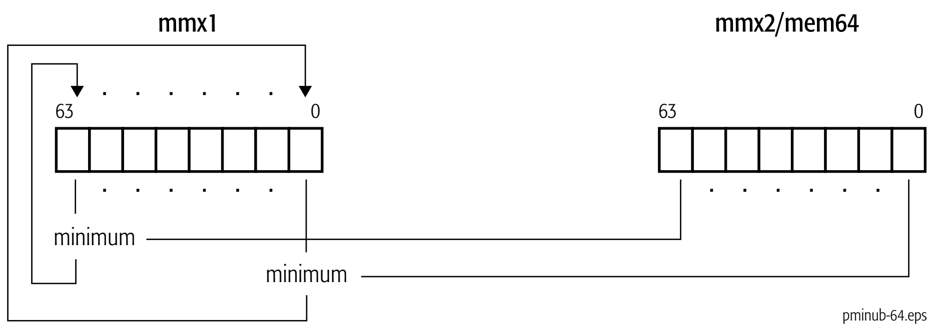
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The SSE instructions are not supported, as indicated by bit 25 in CPUID extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PMINUB Packed Minimum Unsigned Bytes

The PMINUB instruction compares each of the packed 8-bit unsigned integer values in the first source operand with the corresponding packed 8-bit unsigned integer value in the second source operand and writes the minimum of the two values for each comparison in the corresponding byte of the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PMINUB *mmx1*, *mmx2/mem64*	0F DA /*r*	Compares packed unsigned 8-bit integer values in an MMX™ register and another MMX™ register or 64-bit memory location and writes the minimum value of each comparison in the destination MMX™ register.



Related Instructions

PMAXSW, PMAXUB, PMINSW

rFLAGS Affected

None

Exceptions

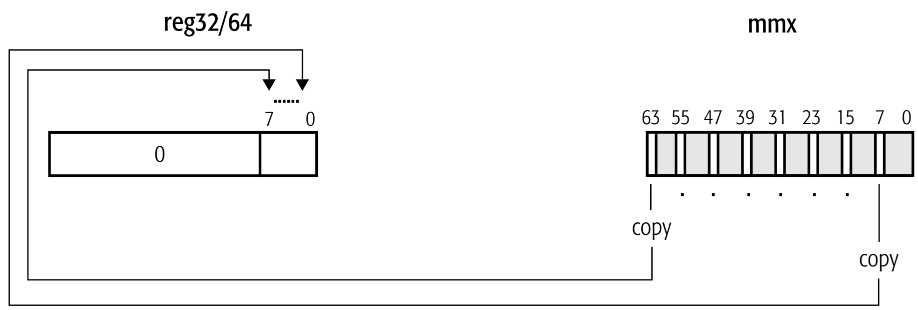
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The SSE instructions are not supported, as indicated by bit 25 in CPUID extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PMOVMSKB Packed Move Mask Byte

The PMOVMSKB instruction moves the most-significant bit of each byte in the source operand to the destination, with zero-extension to 32 or 64 bits. The destination and source operands are a 32-bit or 64-bit general-purpose register and an MMX register.

If the source operand is an XMM register, the result is written to the low-order word of the general-purpose register. If the source operand is an MMX register, the result is written to the low-order byte of the general-purpose register.

Mnemonic	Opcode	Description
PMOVMSKB *reg32/64*, *mmx*	0F D7 /*r*	Moves most-significant bit of each byte in an MMX™ register to the low-order byte of a 32-bit or 64-bit general-purpose register.



Related Instructions

MOVMSKPD, MOVMSKPS

rFLAGS Affected

None

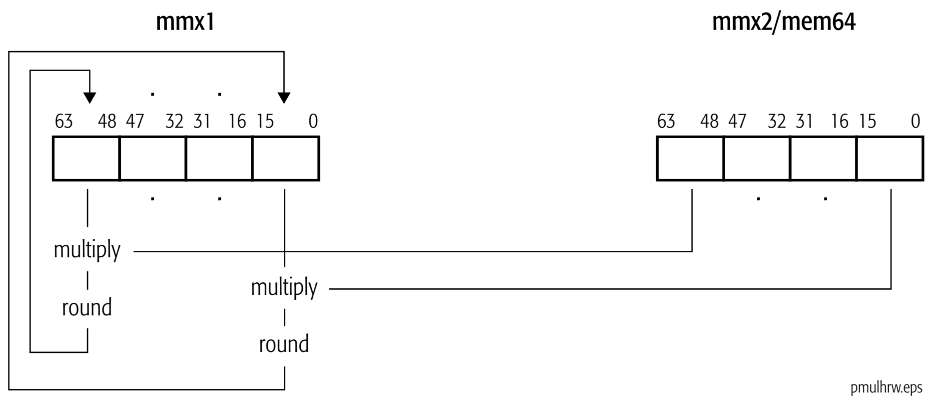
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The SSE instructions are not supported, as indicated by bit 25 in CPUID extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.

PMULHRW Packed Multiply High Rounded Word

The PMULHRW instruction multiplies each of the four packed 16-bit signed integer values in the first source operand by the corresponding packed 16-bit integer value in the second source operand, adds 8000h to the lower 16 bits of the intermediate 32-bit result of each multiplication, and writes the high-order 16 bits of each result in the corresponding word of the destination (first source) The addition of 8000h results in the rounding of the result, providing a numerically more accurate result than the PMULHW instruction, which truncates the result. The first source/destination operand is an MMX register. The second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PMULHRW *mmx1*, *mmx2/mem64*	0F 0F B7	Multiply 16-bit signed integer values in an MMX™ register and another MMX™ register or 64-bit memory location and write rounded result in the destination MMX™ register.



Related Instructions

None

rFLAGS Affected

None

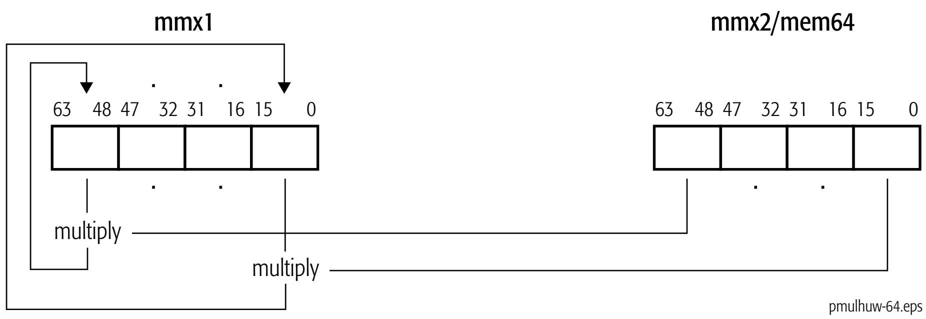
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The AMD 3DNow!™ instructions are not supported, as indicated by bit 31 in CPUID extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PMULHUW Packed Multiply High Unsigned Word

The PMULHUW instruction multiplies each packed unsigned 16-bit values in the first source operand by the corresponding packed unsigned word in the second source operand and writes the high-order 16 bits of each intermediate 32-bit result in the corresponding word of the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	**Opcode**	**Description**
PMULHUW *mmx1*, *mmx2/mem64*	0F E4 /*r*	Multiplies packed 16-bit values in an MMX™ register by the packed 16-bit values in another MMX™ register or 64-bit memory location and writes the high-order 16 bits of each result in the destination MMX™ register.



Related Instructions

PMADDWD, PMULHW, PMULLW, PMULUDQ

rFLAGS Affected

None

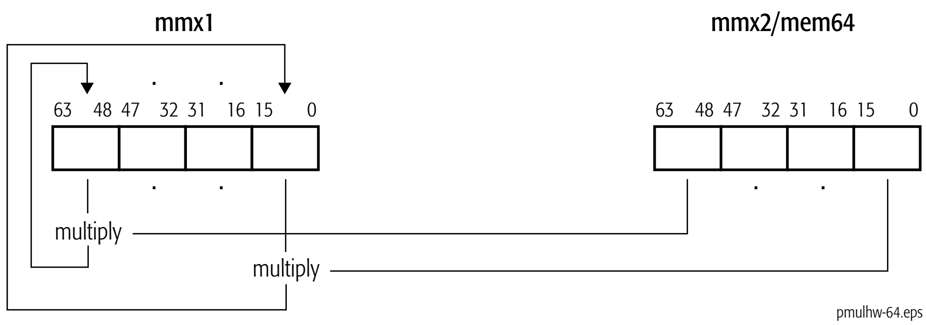
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The SSE instructions are not supported, as indicated by bit 25 in CPUID extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PMULHW Packed Multiply High Signed Word

The PMULHW instruction multiplies each packed 16-bit signed integer value in the first source operand by the corresponding packed 16-bit signed integer in the second source operand and writes the high-order 16 bits of the intermediate 32-bit result of each multiplication in the corresponding word of the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PMULHW *mmx1*, *mmx2/mem64*	0F E5 */r*	Multiplies packed 16-bit signed integer values in an MMX™ register and another MMX™ register or 64-bit memory location and writes the high-order 16 bits of each result in the destination MMX™ register.



Related Instructions

PMADDWD, PMULHUW, PMULLW, PMULUDQ

rFLAGS Affected

None

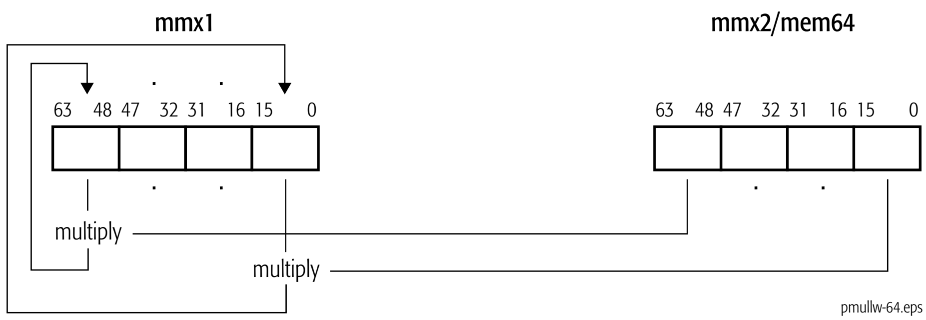
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The MMX instructions are not supported, as indicated by bit 23 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PMULLW Packed Multiply Low Signed Word

The PMULLW instruction multiplies each packed 16-bit signed integer value in the first source operand by the corresponding packed 16-bit signed integer in the second source operand and writes the low-order 16 bits of the intermediate 32-bit result of each multiplication in the corresponding word of the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PMULLW *mmx1*, *mmx2/mem64*	0F D5 /r	Multiplies packed 16-bit signed integer values in an MMX™ register and another MMX™ register or 64-bit memory location and writes the low-order 16 bits of each result in the destination MMX™ register.



Related Instructions

PMADDWD, PMULHUW, PMULHW, PMULUDQ

rFLAGS Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The MMX instructions are not supported, as indicated by bit 23 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PMULUDQ Packed Multiply Unsigned Doubleword and Store Quadword

The PMULUDQ instruction multiplies two 32-bit unsigned integer values in the low-order doubleword of the first and second source operands and writes the 64-bit result in the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PMULUDQ *mmx1*, *mmx2/mem64*	0F F4 /*r*	Multiplies low-order 32-bit unsigned integer value in an MMX™ register and another MMX™ register or 64-bit memory location and writes the 64-bit result in the destination MMX™ register.

Related Instructions

PMADDWD, PMULHUW, PMULHW, PMULLW

rFLAGS Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The SSE2 instructions are not supported, as indicated by bit 26 in CPUID extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

POR Packed Logical Bitwise OR

The POR instruction performs a bitwise logical OR of the values in the first and second source operands and writes the result in the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
POR *mmx1*, *mmx2/mem64*	0F EB */r*	Performs bitwise logical OR of values in an MMX™ register and in another MMX™ register or 64-bit memory location and writes the result in the destination MMX™ register.

Related Instructions

PAND, PANDN, PXOR

rFLAGS Affected

None

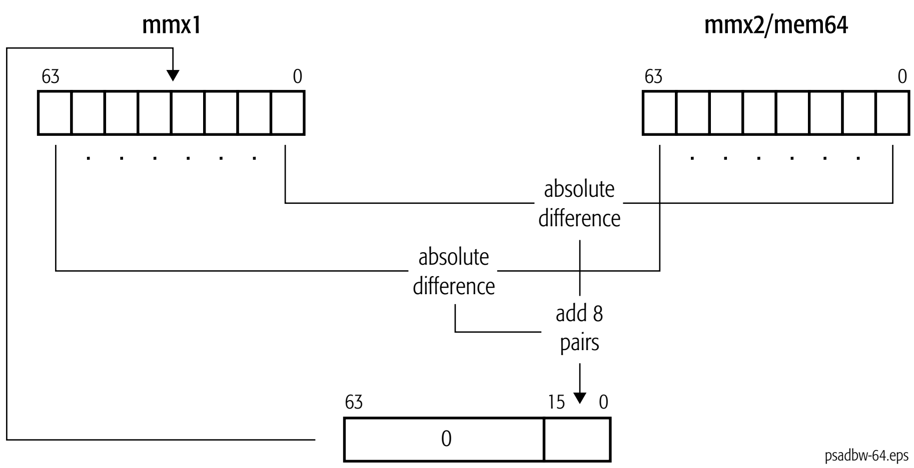
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The MMX instructions are not supported, as indicated by bit 23 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PSADBW Packed Sum of Absolute Differences of Bytes Into a Word

The PSADBW instruction computes the absolute differences of eight corresponding packed 8-bit unsigned integers in the first and second source operands and writes the unsigned 16-bit integer result of the sum of the eight differences in a word in the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location. The result is stored in the low-order word of the destination operand, and the remaining bytes in the destination are cleared to all 0s.

Mnemonic	Opcode	Description
PSADBW *mmx1*, *mmx2/mem64*	0F F6 /r	Compute the sum of the absolute differences of packed 8-bit unsigned integer values in an MMX™ register and another MMX™ register or 64-bit memory location and writes the 16-bit unsigned integer result in the destination MMX™ register.



rFLAGS Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The SSE instructions are not supported, as indicated by bit 25 in CPUID extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PSHUFW Packed Shuffle Words

The PSHUFW instruction moves any one of the four packed words in an MMX register or 64-bit memory location to a specified word location in another MMX register. In each case, the selection of the value of the destination word is determined by a two-bit field in the immediate-byte operand, with bits 0 and 1 selecting the contents of the low-order word, bits 2 and 3 selecting the second word, bits 4 and 5 selecting the third word, and bits 6 and 7 selecting the high-order word. Refer to Table 1-23 on page 173. A word in the source operand may be copied to more than one word in the destination.

Mnemonic	Opcode	Description
PSHUFW *mmx1*, *mmx2/mem64*, *imm8*	0F 70 /*r ib*	Shuffles packed 16-bit values in an MMX™ register or 64-bit memory location and puts the result in another XMM register.

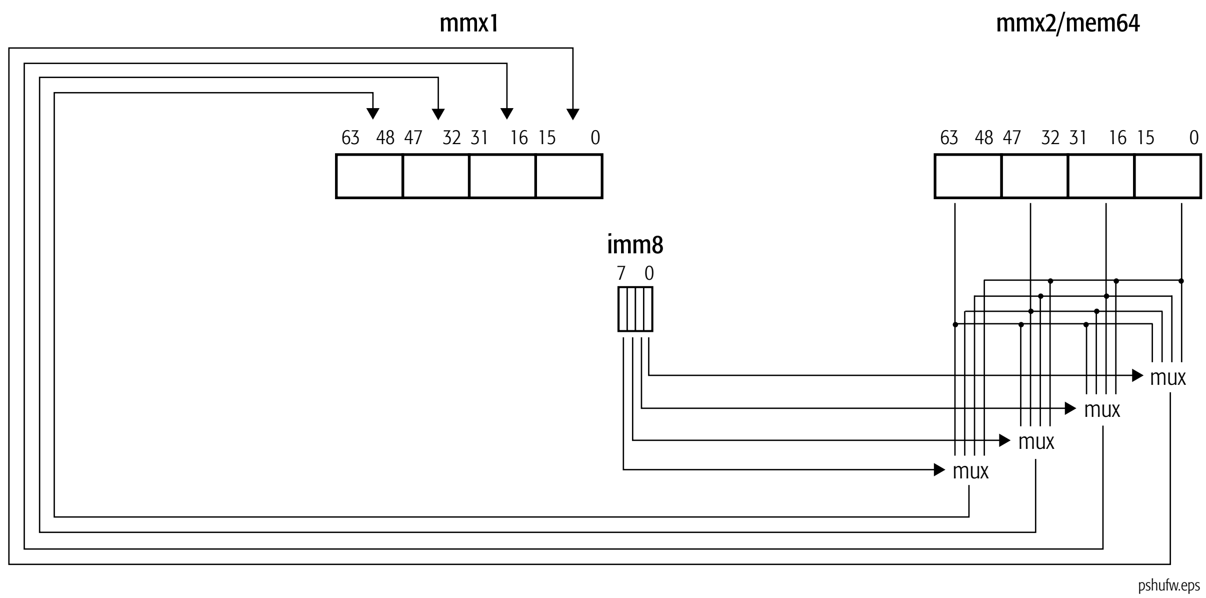


Table 1-23. Immediate-Byte Operand Encoding for PSHUFW

Immediate-Byte Bit Field	Value of Bit Field	Destination Bits Filled	Source Bits Moved
1–0	0	15–0	15–0
	1	15–0	31–16
	2	15–0	47–32
	3	15–0	63–48
3–2	0	31–16	15–0
	1	31–16	31–16
	2	31–16	47–32
	3	31–16	63–48
5–4	0	47–32	15–0
	1	47–32	31–16
	2	47–32	47–32
	3	47–32	63–48
7–6	0	63–48	15–0
	1	63–48	31–16
	2	63–48	47–32
	3	63–48	63–48

Related Instructions

PSHUFD, PSHUFHW, PSHUFLW

rFLAGS Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The SSE instructions are not supported, as indicated by bit 25 in CPUID extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

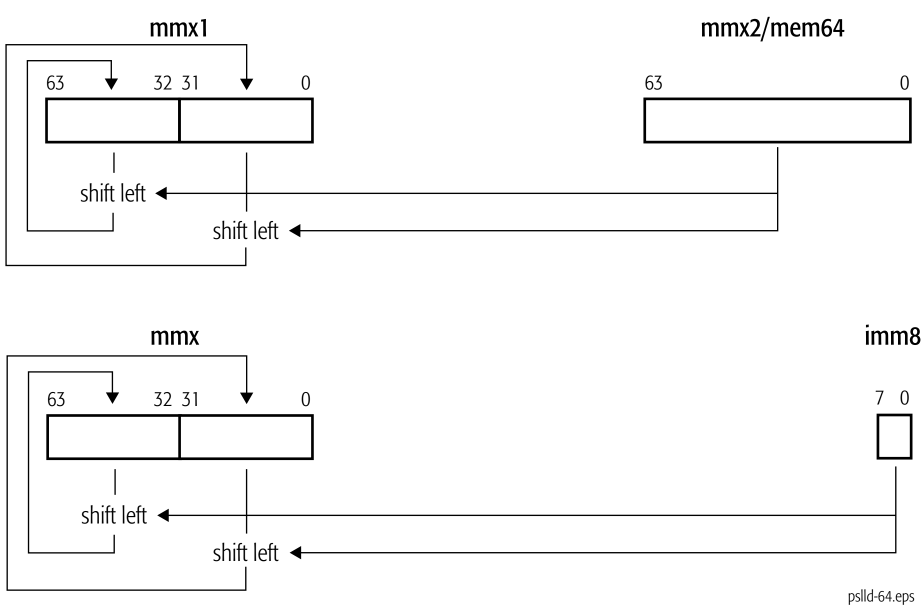
PSLLD Packed Shift Left Logical Doublewords

The PSLLD instruction left-shifts each of the packed 32-bit values in the first source operand by the number of bits specified in the second source operand and writes each shifted value in the corresponding doubleword of the destination (first source). The first source/destination and second source operands are:

- an MMX register and another MMX register or 64-bit memory location, or
- an MMX register and an immediate byte value.

The low-order bits that are emptied by the shift operation are cleared to 0. If the shift value is greater than 31, the destination is cleared to all 0s.

Mnemonic	Opcode	Description
PSLLD *mmx1*, *mmx2/mem64*	0F F2 /*r*	Left-shifts packed doublewords in an MMX™ register by the amount specified in an MMX™ register or 64-bit memory location.
PSLLD *mmx*, *imm8*	0F 72 /6 *ib*	Left-shifts packed doublewords in an MMX™ register by the amount specified in an immediate byte value.



Related Instructions

PSLLDQ, PSLLQ, PSLLW, PSRAD, PSRAW, PSRLD, PSRLDQ, PSRLQ, PSRLW

rFLAGS Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The MMX instructions are not supported, as indicated by bit 23 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

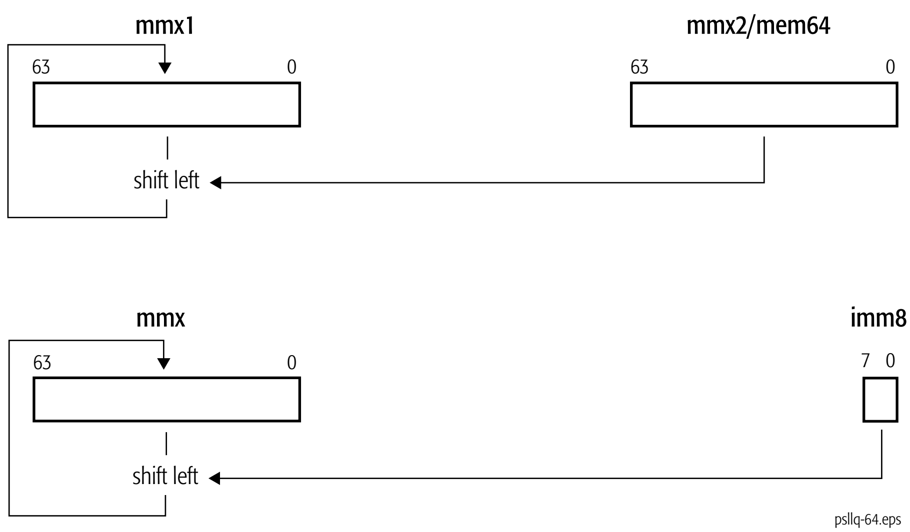
PSLLQ Packed Shift Left Logical Quadwords

The PSLLQ instruction left-shifts each 64-bit value in the first source operand by the number of bits specified in the second source operand and writes each shifted value in the corresponding quadword of the destination (first source). The first source/destination and second source operands are:

- an MMX register and another MMX register or 64-bit memory location, or
- an MMX register and an immediate byte value.

The low-order bits that are emptied by the shift operation are cleared to 0. If the shift value is greater than 63, the destination is cleared to all 0s.

Mnemonic	Opcode	Description
PSLLQ *mmx1*, *mmx2/mem64*	0F F3 /*r*	Left-shifts quadword in an MMX™ register by the amount specified in an MMX™ register or 64-bit memory location.
PSLLQ *mmx imm8*	0F 73 /6 *ib*	Left-shifts quadword in an MMX™ register by the amount specified in an immediate byte value.



Related Instructions

PSLLD, PSLLDQ, PSLLW, PSRAD, PSRAW, PSRLD, PSRLDQ, PSRLQ, PSRLW

rFLAGS Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The MMX instructions are not supported, as indicated by bit 23 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

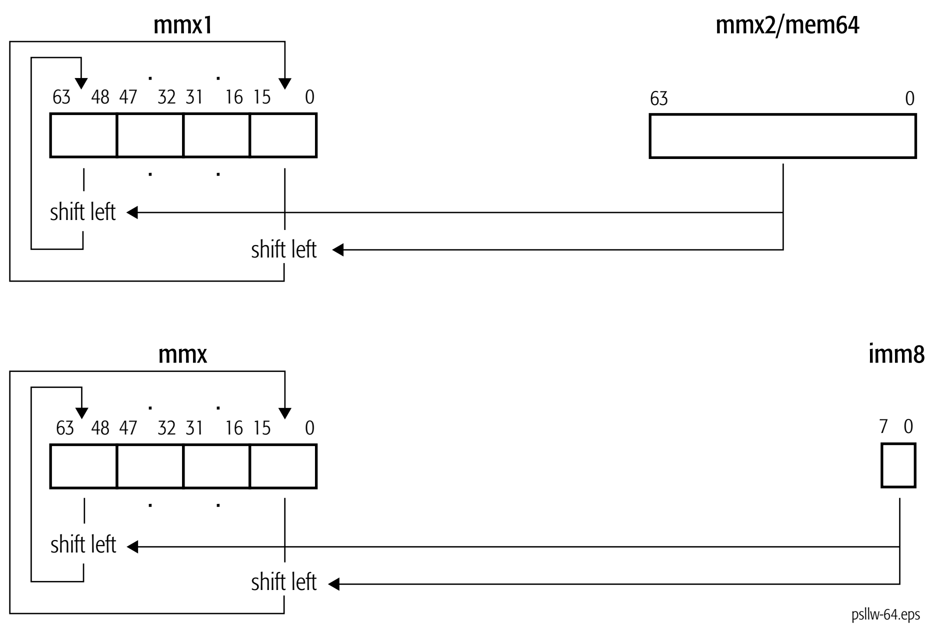
PSLLW Packed Shift Left Logical Words

The PSLLW instruction left-shifts each of the packed 16-bit values in the first source operand by the number of bits specified in the second source operand and writes each shifted value in the corresponding word of the destination (first source). The first source/destination and second source operands are:

- an MMX register and another MMX register or 64-bit memory location, or
- an MMX register and an immediate byte value.

The low-order bits that are emptied by the shift operation are cleared to 0. If the shift value is greater than 15, the destination is cleared to all 0s.

Mnemonic	Opcode	Description
PSLLW *mmx1*, *mmx2/mem64*	0F F1 */r*	Left-shifts packed words in an MMX™ register by the amount specified in an MMX™ register or 64-bit memory location.
PSLLW *mmx, imm8*	0F 71 /6 *ib*	Left-shifts packed words in an MMX™ register by the amount specified in an immediate byte value.



Related Instructions

PSLLD, PSLLDQ, PSLLQ, PSRAD, PSRAW, PSRLD, PSRLDQ, PSRLQ, PSRLW

rFLAGS Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The MMX instructions are not supported, as indicated by bit 23 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

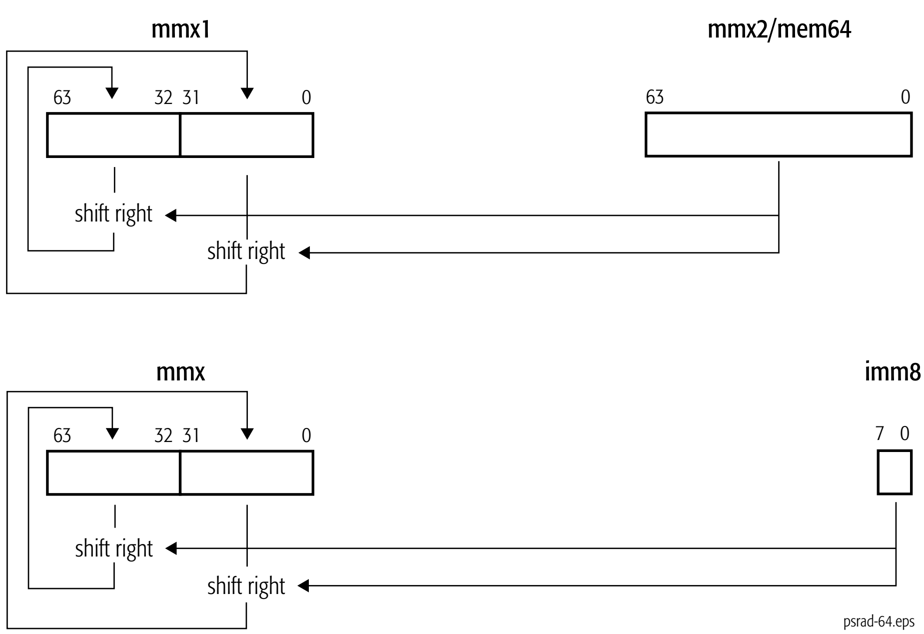
PSRAD Packed Shift Right Arithmetic Doublewords

The PSRAD instruction right-shifts each of the packed 32-bit values in the first source operand by the number of bits specified in the second source operand and writes each shifted value in the corresponding doubleword of the destination (first source). The first source/destination and second source operands are:

- an MMX register and another MMX register or 64-bit memory location, or
- an MMX register and an immediate byte value.

The high-order bits that are emptied by the shift operation are filled with the sign bit of the doubleword's initial value. If the shift value is greater than 31, each doubleword in the destination is filled with the sign bit of the doubleword's initial value.

Mnemonic	Opcode	Description
PSRAD *mmx1*, *mmx2/mem64*	0F E2 /*r*	Right-shifts packed doublewords in an MMX™ register by the amount specified in an MMX™ register or 64-bit memory location.
PSRAD *mmx, imm8*	0F 72 /4 *ib*	Right-shifts packed doublewords in an MMX™ register by the amount specified in an immediate byte value.



Related Instructions

PSLLD, PSLLDQ, PSLLQ, PSLLW, PSRAW, PSRLD, PSRLDQ, PSRLQ, PSRLW

rFLAGS Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The MMX instructions are not supported, as indicated by bit 23 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

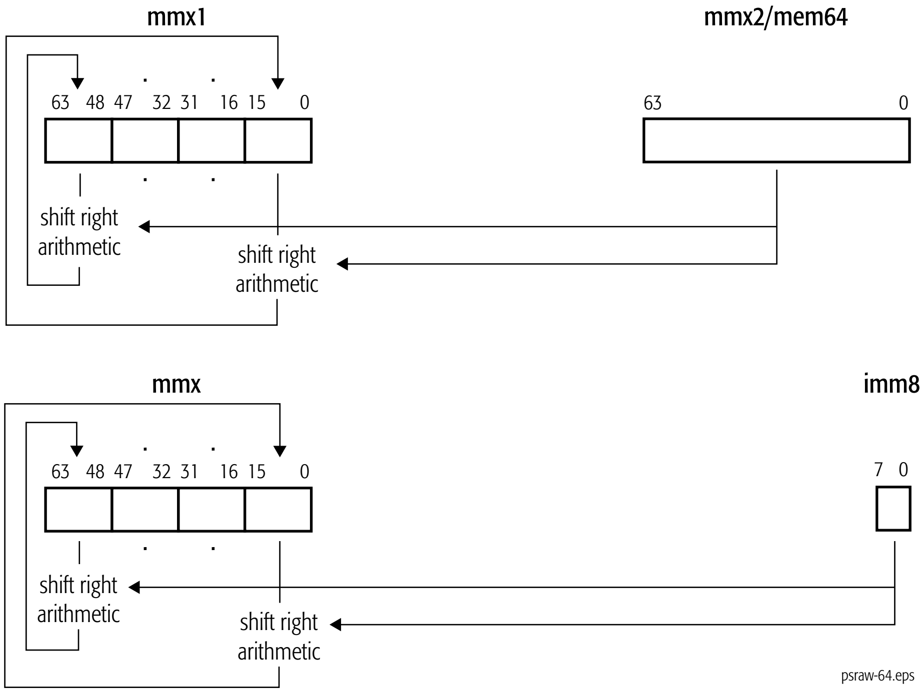
PSRAW Packed Shift Right Arithmetic Words

The PSRAW instruction right-shifts each of the packed 16-bit values in the first source operand by the number of bits specified in the second source operand and writes each shifted value in the corresponding word of the destination (first source). The first source/destination and second source operands are:

- an MMX register and another MMX register or 64-bit memory location, or
- an MMX register and an immediate byte value.

The high-order bits that are emptied by the shift operation are filled with the sign bit of the word's initial value. If the shift value is greater than 15, each word in the destination is filled with the sign bit of the word's initial value.

Mnemonic	Opcode	Description
PSRAW *mmx1*, *mmx2/mem64*	0F E1 */r*	Right-shifts packed words in an MMX™ register by the amount specified in an MMX™ register or 64-bit memory location.
PSRAW *mmx, imm8*	0F 71 /4 *ib*	Right-shifts packed words in an MMX™ register by the amount specified in an immediate byte value.



Related Instructions

PSLLD, PSLLDQ, PSLLQ, PSLLW, PSRAD, PSRLD, PSRLDQ, PSRLQ, PSRLW

rFLAGS Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The MMX instructions are not supported, as indicated by bit 23 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

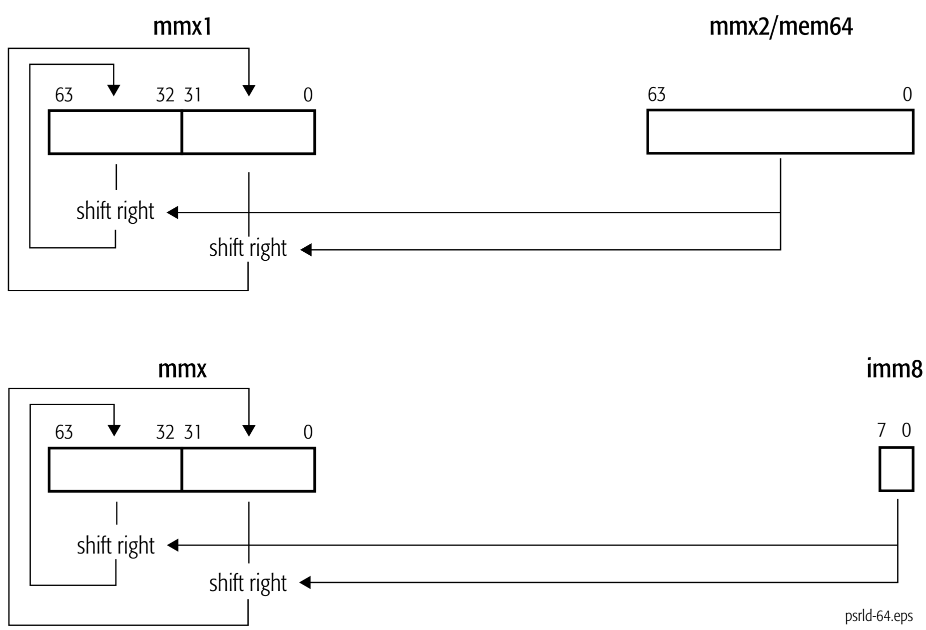
PSRLD Packed Shift Right Logical Doublewords

The PSRLD instruction right-shifts each of the packed 32-bit values in the first source operand by the number of bits specified in the second source operand and writes each shifted value in the corresponding doubleword of the destination (first source). The first source/destination and second source operands are:

- an MMX register and another MMX register or 64-bit memory location, or
- an MMX register and an immediate byte value.

The high-order bits that are emptied by the shift operation are cleared to 0. If the shift value is greater than 31, the destination is cleared to 0.

Mnemonic	Opcode	Description
PSRLD *mmx1*, *mmx2/mem64*	0F D2 /*r*	Right-shifts packed doublewords in an MMX™ register by the amount specified in an MMX™ register or 64-bit memory location.
PSRLD *mmx, imm8*	0F 72 /2 *ib*	Right-shifts packed doublewords in an MMX™ register by the amount specified in an immediate byte value.



Related Instructions

PSLLD, PSLLDQ, PSLLQ, PSLLW, PSRAD, PSRAW, PSRLDQ, PSRLQ, PSRLW

rFLAGS Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The MMX instructions are not supported, as indicated by bit 23 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

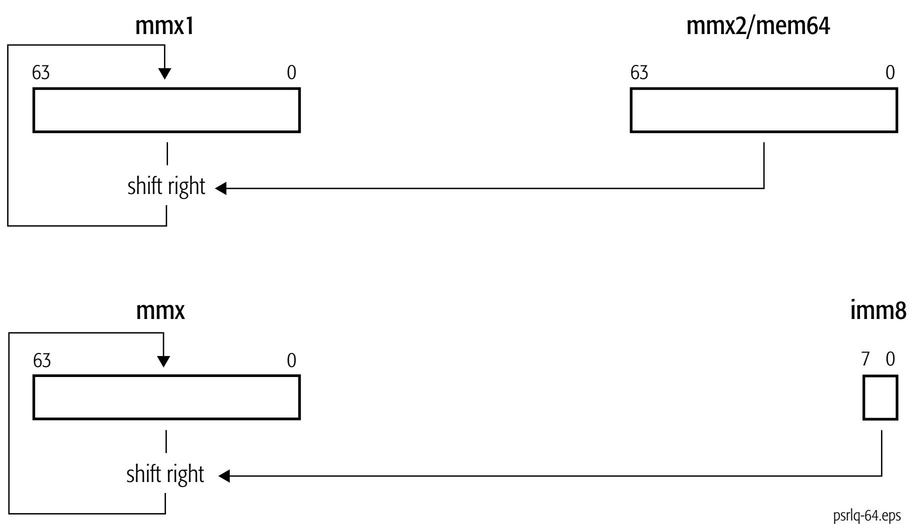
PSRLQ Packed Shift Right Logical Quadwords

The PSRLQ instruction right-shifts each 64-bit value in the first source operand by the number of bits specified in the second source operand and writes each shifted value in the corresponding quadword of the destination (first source). The first source/destination and second source operands are:

- an MMX register and another MMX register or 64-bit memory location, or
- an MMX register and an immediate byte value.

The high-order bits that are emptied by the shift operation are cleared to 0. If the shift value is greater than 63, the destination is cleared to 0.

Mnemonic	Opcode	Description
PSRLQ *mmx1*, *mmx2/mem64*	0F D3 /*r*	Right-shifts quadword in an MMX™ register by the amount specified in an MMX™ register or 64-bit memory location.
PSRLQ *mmx*, *imm8*	0F 73 /2 *ib*	Right-shifts quadword in an MMX™ register by the amount specified in an immediate byte value.



Related Instructions

PSLLD, PSLLDQ, PSLLQ, PSLLW, PSRAD, PSRAW, PSRLD, PSRLDQ, PSRLW

rFLAGS Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The MMX instructions are not supported, as indicated by bit 23 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

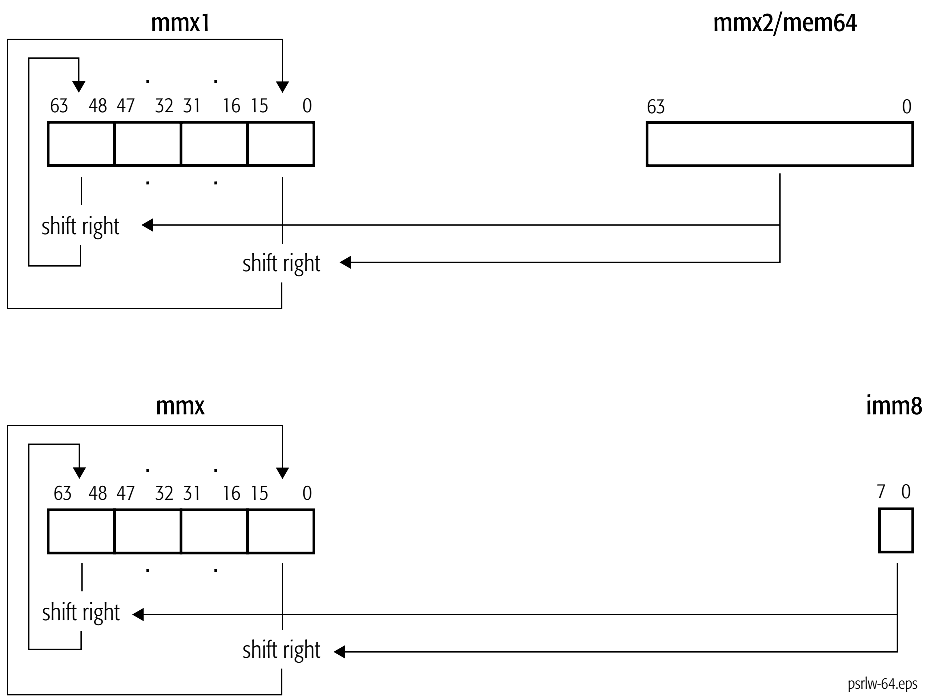
PSRLW Packed Shift Right Logical Words

The PSRLW instruction right-shifts each of the packed 16-bit values in the first source operand by the number of bits specified in the second operand and writes each shifted value in the corresponding word of the destination (first source). The first source/destination and second source operands are:

- an MMX register and another MMX register or 64-bit memory location, or
- an MMX register and an immediate byte value.

The high-order bits that are emptied by the shift operation are cleared to 0. If the shift value is greater than 15, the destination is cleared to 0.

Mnemonic	Opcode	Description
PSRLW *mmx1*, *mmx2/mem64*	0F D1 /*r*	Right-shifts packed words in an MMX™ register by the amount specified in an MMX™ register or 64-bit memory location.
PSRLW *mmx*, *imm8*	0F 71 /2 *ib*	Right-shifts packed words in an MMX™ register by the amount specified in an immediate byte value.



Related Instructions

PSLLD, PSLLDQ, PSLLQ, PSLLW, PSRAD, PSRAW, PSRLD, PSRLDQ, PSRLQ

rFLAGS Affected

None

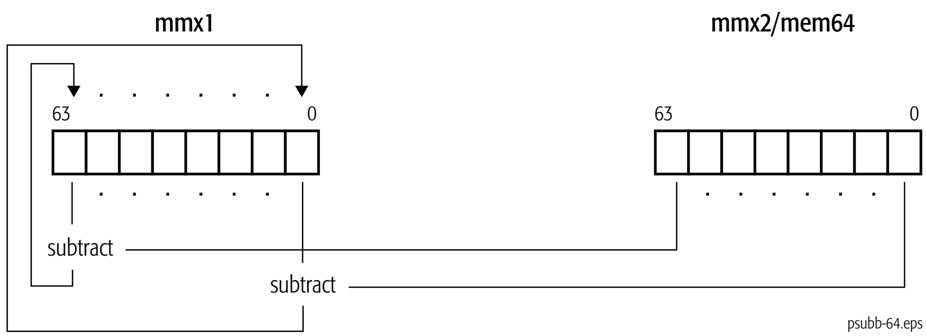
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The MMX instructions are not supported, as indicated by bit 23 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PSUBB Packed Subtract Bytes

The PSUBB instruction subtracts each packed 8-bit integer value in the second source operand from the corresponding packed 8-bit integer in the first source operand and writes the integer result of each subtraction in the corresponding byte of the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PSUBB *mmx1*, *mmx2/mem64*	0F F8 */r*	Subtracts packed byte integer values in an MMX™ register or 64-bit memory location from packed byte integer values in another MMX™ register and writes the result in the destination MMX™ register.



This instruction operates on both signed and unsigned integers. If the result overflows, the carry is ignored (neither the overflow nor carry bit in rFLAGS is set), and only the low-order 8 bits of each result are written in the destination.

Related Instructions

PSUBD, PSUBQ, PSUBSB, PSUBSW, PSUBUSB, PSUBUSW, PSUBW

rFLAGS Affected

None

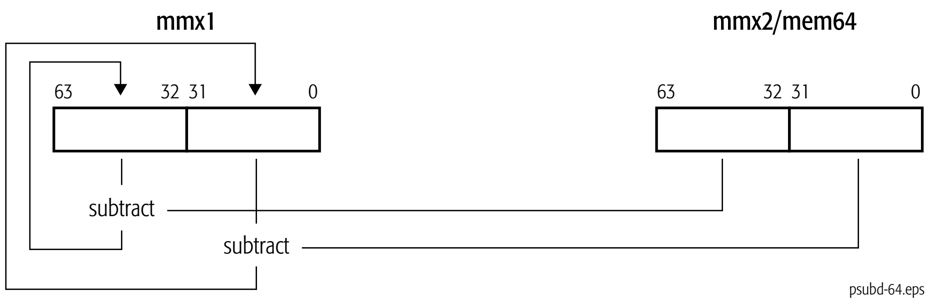
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The MMX instructions are not supported, as indicated by bit 23 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PSUBD Packed Subtract Doublewords

The PSUBD instruction subtracts each packed 32-bit integer value in the second source operand from the corresponding packed 32-bit integer in the first source operand and writes the integer result of each subtraction in the corresponding doubleword of the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PSUBD *mmx1*, *mmx2/mem64*	0F FA /*r*	Subtracts packed 32-bit integer values in an MMX™ register or 64-bit memory location from packed 32-bit integer values in another MMX™ register and writes the result in the destination MMX™ register.



This instruction operates on both signed and unsigned integers. If the result overflows, the carry is ignored (neither the overflow nor carry bit in rFLAGS is set), and only the low-order 32 bits of each result are written in the destination.

Related Instructions

PSUBB, PSUBQ, PSUBSB, PSUBSW, PSUBUSB, PSUBUSW, PSUBW

rFLAGS Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The MMX instructions are not supported, as indicated by bit 23 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PSUBQ Packed Subtract Quadword

The PSUBQ instruction subtracts each packed 64-bit integer value in the second source operand from the corresponding packed 64-bit integer in the first source operand and writes the integer result of each subtraction in the corresponding quadword of the destination (first source). The first source/destination and source operands are an MMX register and another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PSUBQ *mmx1*, *mmx2/mem64*	0F FB /r	Subtracts packed 64-bit integer values in an MMX™ register or 64-bit memory location from packed 64-bit integer values in another MMX™ register and writes the result in the destination MMX™ register.

This instruction operates on both signed and unsigned integers. If the result overflows, the carry is ignored (neither the overflow nor carry bit in rFLAGS is set), and only the low-order 64 bits of each result are written in the destination.

Related Instructions

PSUBB, PSUBD, PSUBSB, PSUBSW, PSUBUSB, PSUBUSW, PSUBW

rFLAGS Affected

None

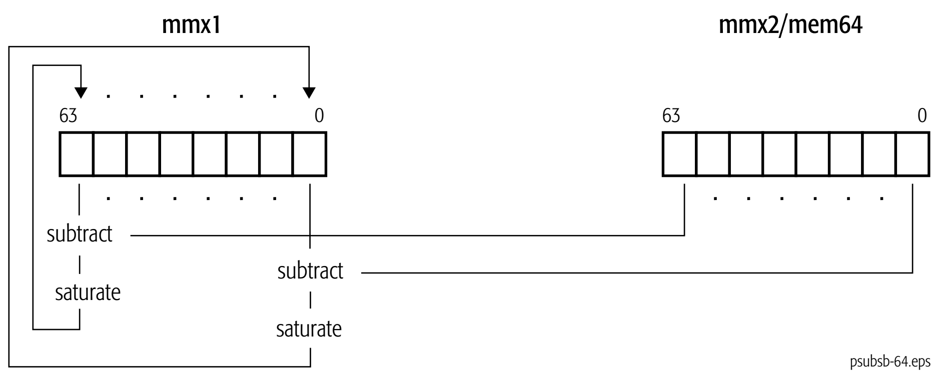
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The SSE2 instructions are not supported, as indicated by bit 26 in CPUID extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PSUBSB — Packed Subtract Signed With Saturation Bytes

The PSUBSB instruction subtracts each packed 8-bit signed integer value in the second source operand from the corresponding packed 8-bit signed integer in the first source operand and writes the signed integer result of each subtraction in the corresponding byte of the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PSUBSB *mmx1*, *mmx2/mem64*	0F E8 /*r*	Subtracts packed byte signed integer values in an MMX™ register or 64-bit memory location from packed byte integer values in another MMX™ register and writes the result in the destination MMX™ register.



For each packed value in the destination, if the value is larger than the largest signed 8-bit integer, it is saturated to 7Fh, and if the value is smaller than the smallest signed 8-bit integer, it is saturated to 80h.

Related Instructions

PSUBB, PSUBD, PSUBQ, PSUBSW, PSUBUSB, PSUBUSW, PSUBW

rFLAGS Affected

None

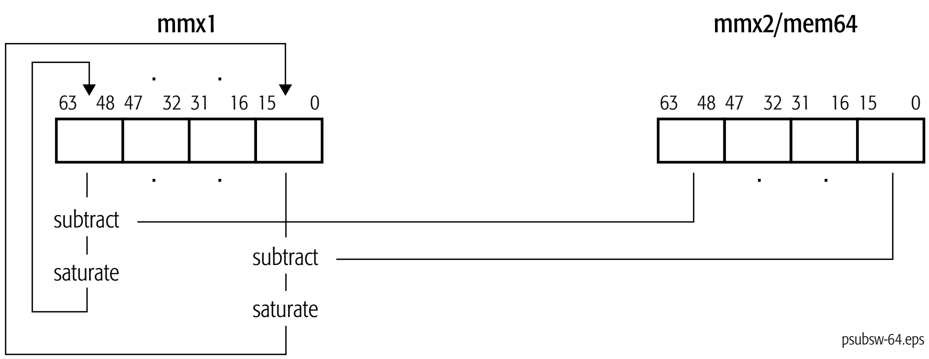
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The MMX instructions are not supported, as indicated by bit 23 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PSUBSW Packed Subtract Signed With Saturation Words

The PSUBSW instruction subtracts each packed 16-bit signed integer value in the second source operand from the corresponding packed 16-bit signed integer in the first source operand and writes the signed integer result of each subtraction in the corresponding word of the destination (first source). The first source/destination and source operands are an MMX register and another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PSUBSW *mmx1*, *mmx2/mem64*	0F E9 */r*	Subtracts packed 16-bit signed integer values in an MMX™ register or 64-bit memory location from packed 16-bit integer values in another MMX™ register and writes the result in the destination MMX™ register.



For each packed value in the destination, if the value is larger than the largest signed 16-bit integer, it is saturated to 7FFFh, and if the value is smaller than the smallest signed 16-bit integer, it is saturated to 8000h.

Related Instructions

PSUBB, PSUBD, PSUBQ, PSUBSB, PSUBUSB, PSUBUSW, PSUBW

rFLAGS Affected

None

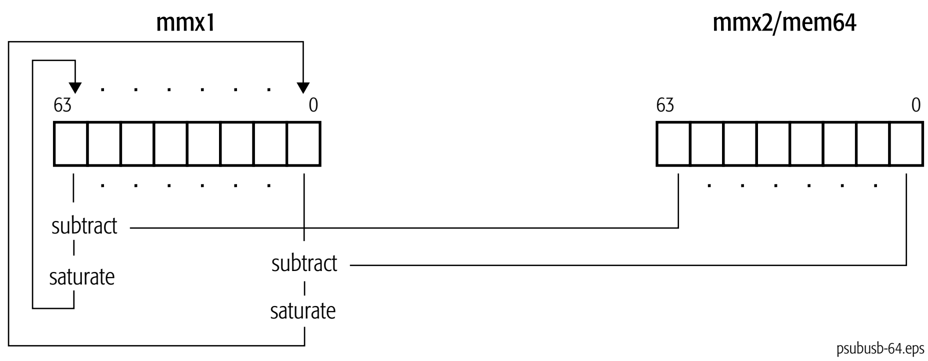
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The MMX instructions are not supported, as indicated by bit 23 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PSUBUSB Packed Subtract Unsigned and Saturate Bytes

The PSUBUSB instruction subtracts each packed 8-bit unsigned integer value in the second source operand from the corresponding packed 8-bit unsigned integer in the first source operand and writes the unsigned integer result of each subtraction in the corresponding byte of the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PSUBUSB *mmx1*, *mmx2/mem64*	0F D8 /r	Subtracts packed byte unsigned integer values in an MMX™ register or 64-bit memory location from packed byte integer values in another MMX™ register and writes the result in the destination MMX™ register.



For each packed value in the destination, if the value is larger than the largest unsigned 8-bit integer, it is saturated to FFh, and if the value is smaller than the smallest unsigned 8-bit integer, it is saturated to 00h.

Related Instructions

PSUBB, PSUBD, PSUBQ, PSUBSB, PSUBSW, PSUBUSW, PSUBW

rFLAGS Affected

None

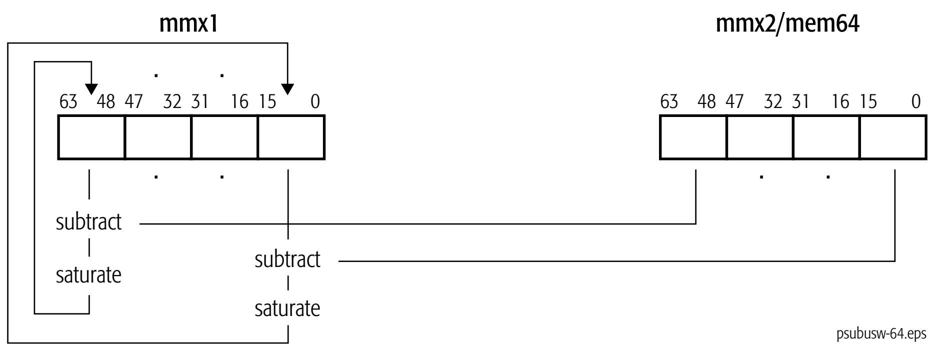
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The MMX instructions are not supported, as indicated by bit 23 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PSUBUSW Packed Subtract Unsigned and Saturate Words

The PSUBUSW instruction subtracts each packed 16-bit unsigned integer value in the second source operand from the corresponding packed 16-bit unsigned integer in the first source operand and writes the unsigned integer result of each subtraction in the corresponding word of the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PSUBUSW *mmx1*, *mmx2/mem64*	0F D9 */r*	Subtracts packed 16-bit unsigned integer values in an MMX™ register or 64-bit memory location from packed 16-bit integer values in another MMX™ register and writes the result in the destination MMX™ register.



For each packed value in the destination, if the value is larger than the largest unsigned 16-bit integer, it is saturated to FFFFh, and if the value is smaller than the smallest unsigned 16-bit integer, it is saturated to 0000h.

Related Instructions

PSUBB, PSUBD, PSUBQ, PSUBSB, PSUBSW, PSUBUSB, PSUBW

rFLAGS Affected

None

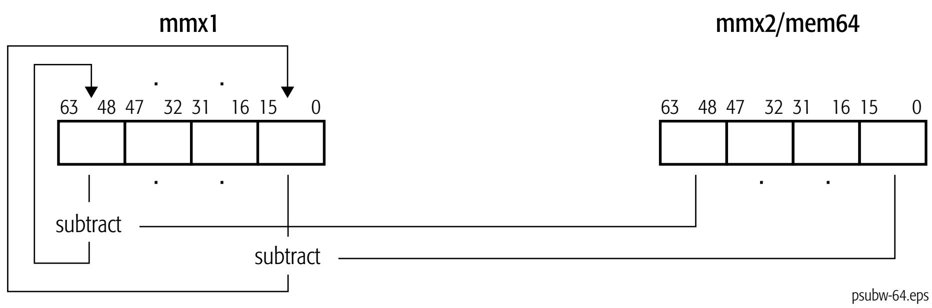
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The MMX instructions are not supported, as indicated by bit 23 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PSUBW Packed Subtract Words

The PSUBW instruction subtracts each packed 16-bit integer value in the second source operand from the corresponding packed 16-bit integer in the first source operand and writes the integer result of each subtraction in the corresponding word of the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PSUBW *mmx1*, *mmx2/mem64*	0F F9 /*r*	Subtracts packed 16-bit integer values in an MMX™ register or 64-bit memory location from packed 16-bit integer values in another MMX™ register and writes the result in the destination MMX™ register.



This instruction operates on both signed and unsigned integers. If the result overflows, the carry is ignored (neither the overflow nor carry bit in rFLAGS is set), and only the low-order 16 bits of the result are written in the destination.

Related Instructions

PSUBB, PSUBD, PSUBQ, PSUBSB, PSUBSW, PSUBUSB, PSUBUSW

rFLAGS Affected

None

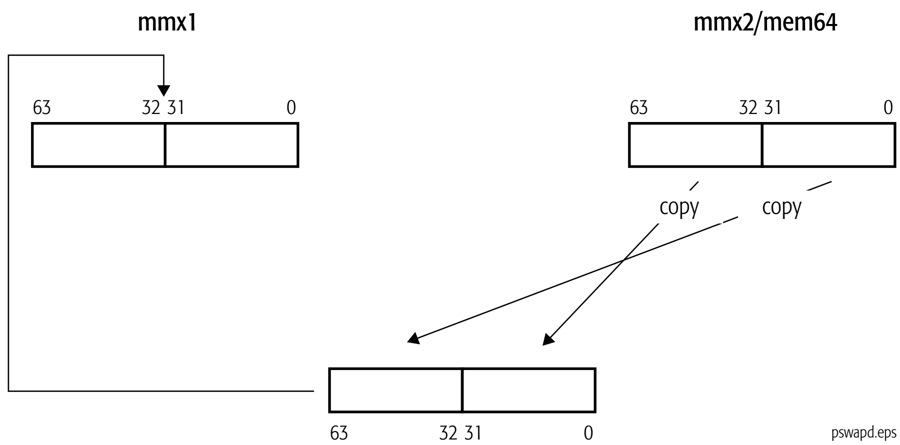
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The MMX instructions are not supported, as indicated by bit 23 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PSWAPD Packed Swap Doubleword

The PSWAPD instruction swaps (reverses) the two packed 32-bit values in the source operand and writes each swapped value in the corresponding doubleword of the destination. The source operand is an MMX register or 64-bit memory location. The destination is another MMX register.

Mnemonic	Opcode	Description
PSWAPD *mmx1*, *mmx2/mem64*	0F 0F BB	Swaps packed 32-bit values in an MMX™ register or 64-bit memory location and writes each value in the destination MMX™ register.



Related Instructions

None

rFLAGS Affected

None

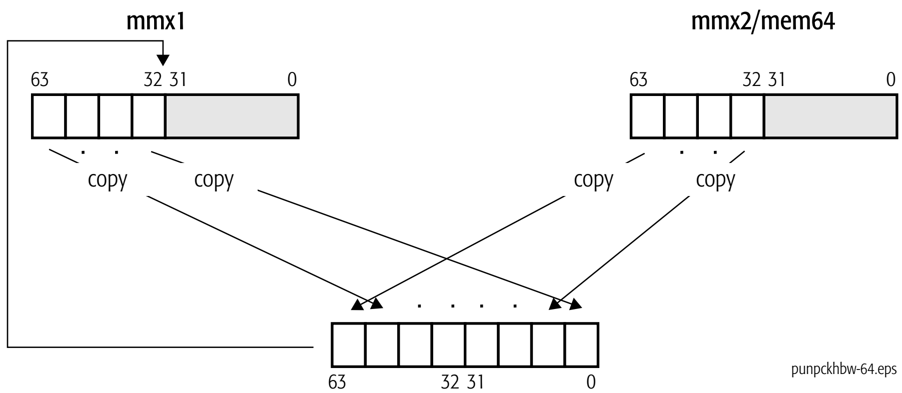
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The AMD Extensions to 3DNow!™ are not supported, as indicated by bit 30 in CPUID extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PUNPCKHBW Unpack and Interleave High Bytes

The PUNPCKHBW instruction unpacks the high-order bytes from the first and second source operands and packs them into interleaved-byte words in the destination (first source). The low-order bytes of the source operands are ignored. The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PUNPCKHBW *mmx1*, *mmx2/mem64*	0F 68 */r*	Unpacks the four high-order bytes in an MMX™ register and another MMX™ register or 64-bit memory location and packs them into interleaved bytes in the destination MMX™ register.



If the second source operand is all 0s, the destination contains the bytes from the first source operand zero-extended to 16 bits. This operation is useful for expanding unsigned 8-bit values to unsigned 16-bit operands for subsequent processing that requires higher precision.

Related Instructions

PUNPCKHDQ, PUNPCKHQDQ, PUNPCKHWD, PUNPCKLBW, PUNPCKLDQ, PUNPCKLQDQ, PUNPCKLWD

rFLAGS Affected

None

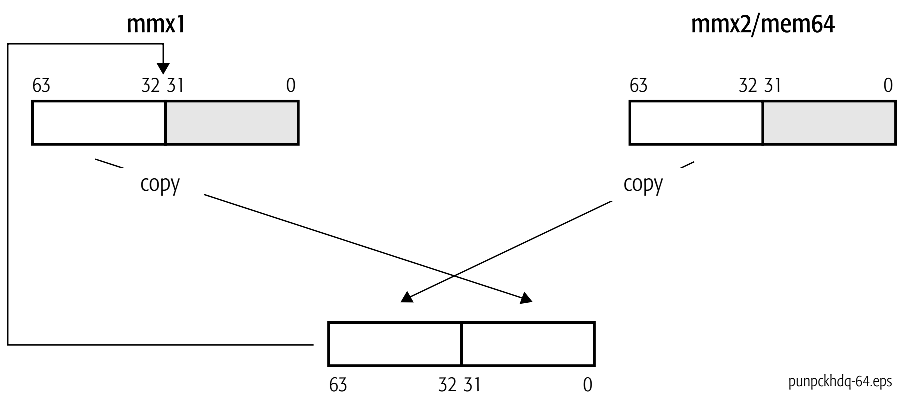
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The MMX instructions are not supported, as indicated by bit 23 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PUNPCKHDQ Unpack and Interleave High Doublewords

The PUNPCKHDQ instruction unpacks the high-order doublewords from the first and second source operands and packs them into interleaved-doubleword quadwords in the destination (first source). The low-order doublewords of the source operands are ignored. The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PUNPCKHDQ *mmx1*, *mmx2/mem64*	0F 6A /*r*	Unpacks the high-order doubleword in an MMX™ register and another MMX™ register or 64-bit memory location and packs them into interleaved doublewords in the destination MMX™ register.



If the second source operand is all 0s, the destination contains the doubleword(s) from the first source operand zero-extended to 64 bits. This operation is useful for expanding unsigned 32-bit values to unsigned 64-bit operands for subsequent processing that requires higher precision.

Related Instructions

PUNPCKHBW, PUNPCKHQDQ, PUNPCKHWD, PUNPCKLBW, PUNPCKLDQ, PUNPCKLQDQ, PUNPCKLWD

rFLAGS Affected

None

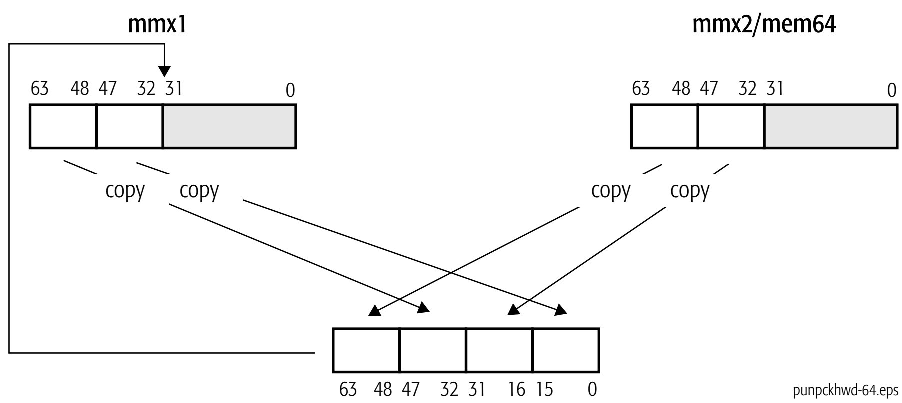
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The MMX instructions are not supported, as indicated by bit 23 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PUNPCKHWD Unpack and Interleave High Words

The PUNPCKHWD instruction unpacks the high-order words from the first and second source operands and packs them into interleaved-word doublewords in the destination (first source). The low-order words of the source operands are ignored. The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PUNPCKHWD *mmx1*, *mmx2/mem64*	0F 69 /*r*	Unpacks two high-order words in an MMX™ register and another MMX™ register or 64-bit memory location and packs them into interleaved words in the destination MMX™ register.



If the second source operand is all 0s, the destination contains the words from the first source operand zero-extended to 32 bits. This operation is useful for expanding unsigned 16-bit values to unsigned 32-bit operands for subsequent processing that requires higher precision.

Related Instructions

PUNPCKHBW, PUNPCKHDQ, PUNPCKHQDQ, PUNPCKLBW, PUNPCKLDQ, PUNPCKLQDQ, PUNPCKLWD

rFLAGS Affected

None

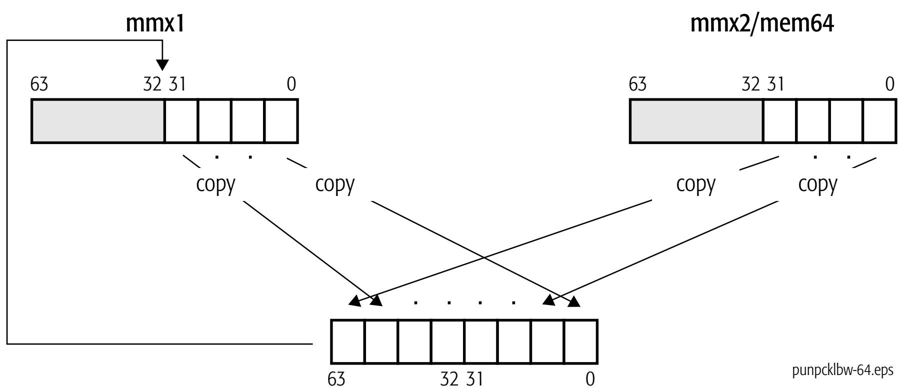
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The MMX instructions are not supported, as indicated by bit 23 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PUNPCKLBW Unpack and Interleave Low Bytes

The PUNPCKLBW instruction unpacks the low-order bytes from the first and second source operands and packs them into interleaved-byte words in the destination (first source). The high-order bytes of the source operands are ignored. The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PUNPCKLBW *mmx1*, *mmx2/mem64*	0F 60 /*r*	Unpacks the four low-order bytes in an MMX™ register and another MMX™ register or 64-bit memory location and packs them into interleaved bytes in the destination MMX™ register.



If the second source operand is all 0s, the destination contains the bytes from the first source operand zero-extended to 16 bits. This operation is useful for expanding unsigned 8-bit values to unsigned 16-bit operands for subsequent processing that requires higher precision.

Related Instructions

PUNPCKHBW, PUNPCKHDQ, PUNPCKHQDQ, PUNPCKHWD, PUNPCKLDQ, PUNPCKLQDQ, PUNPCKLWD

rFLAGS Affected

None

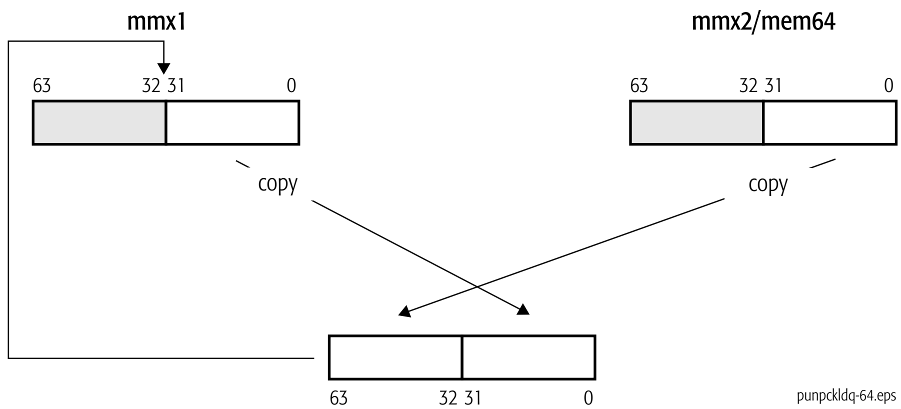
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The MMX instructions are not supported, as indicated by bit 23 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PUNPCKLDQ Unpack and Interleave Low Doublewords

The PUNPCKLDQ instruction unpacks the low-order doublewords from the first and second source operands and packs them into interleaved-doubleword quadwords in the destination (first source). The high-order doublewords of the source operands are ignored. The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PUNPCKLDQ *mmx1*, *mmx2/mem64*	0F 62 /*r*	Unpacks the low-order doubleword in an MMX™ register and another MMX™ register or 64-bit memory location and packs them into interleaved doublewords in the destination MMX™ register.



If the second source operand is all 0s, the destination contains the doubleword(s) from the first source operand zero-extended to 64 bits. This operation is useful for expanding unsigned 32-bit values to unsigned 64-bit operands for subsequent processing that requires higher precision.

Related Instructions

PUNPCKHBW, PUNPCKHDQ, PUNPCKHQDQ, PUNPCKHWD, PUNPCKLBW, PUNPCKLQDQ, PUNPCKLWD

rFLAGS Affected

None

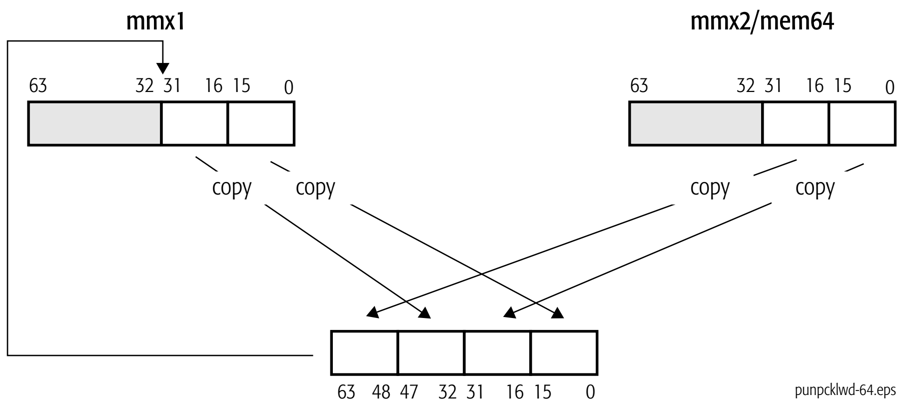
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The MMX instructions are not supported, as indicated by bit 23 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PUNPCKLWD Unpack and Interleave Low Words

The PUNPCKLWD instruction unpacks the low-order words from the first and second source operands and packs them into interleaved-word doublewords in the destination (first source). The high-order words of the source operands are ignored. The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PUNPCKLWD *mmx1*, *mmx2/mem64*	0F 61 /*r*	Unpacks the two low-order words in an MMX™ register and another MMX™ register or 64-bit memory location and packs them into interleaved words in the destination MMX™ register.



If the second source operand is all 0s, the destination contains the words from the first source operand zero-extended to 32 bits. This operation is useful for expanding unsigned 16-bit values to unsigned 32-bit operands for subsequent processing that requires higher precision.

Related Instructions

PUNPCKHBW, PUNPCKHDQ, PUNPCKHQDQ, PUNPCKHWD, PUNPCKLBW, PUNPCKLDQ, PUNPCKLQDQ

rFLAGS Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The MMX instructions are not supported, as indicated by bit 23 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

PXOR Packed Logical Bitwise Exclusive OR

The PXOR instruction performs a bitwise exclusive OR of the values in the first and second source operands and writes the result in the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PXOR *mmx1*, *mmx2/mem64*	0F EF /r	Performs bitwise logical XOR of values in an MMX™ register and in another MMX™ register or 64-bit memory location and writes the result in the destination MMX™ register.

Related Instructions

PAND, PANDN, POR

rFLAGS Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 was set to 1. The MMX instructions are not supported, as indicated by bit 23 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.

2 x87 Floating-Point Instruction Reference

This chapter describes the function, mnemonic syntax, opcodes, condition codes, affected flags, and possible exceptions generated by the x87 floating-point instructions. The x87 floating-point instructions are used in legacy floating-point applications. Most of these instructions load, store, or operate on data located in the x87 ST(0)–ST(7) stack registers (the FPR0–FPR7 physical registers). The remaining instructions within this category are used to manage the x87 floating-point environment.

A given hardware implementation of the x86-64 architecture supports the x87 floating-point instructions if the following CPUID functions are set:

- On-Chip Floating-Point Unit, indicated by bit 0 of CPUID standard function 1 and extended function 8000_0001h.
- CMOV*cc* (conditional moves), indicated by bit 15 of CPUID standard function 1 and extended function 8000_0001h. A 1 in this bit indicates support for x87 floating-point conditional moves (FCMOV*cc*) whenever the On-Chip Floating-Point Unit bit (bit 0) is also 1.

The x87 instructions can be used in legacy mode or long mode. Their use in long mode is available if the following CPUID function bit is set to 1:

- Long Mode, indicated by bit 29 of CPUID extended function 8000_0001h.

Compilation of x87 media programs for execution in 64-bit mode offers two primary advantages: access to the 64-bit virtual address space and access to the RIP-relative addressing mode.

For further information about the x87 floating-point instructions and register resources, see:

- “x87 Floating-Point Programming” in volume 1.
- “Summary of Registers and Data Types” in volume 3.
- “Notation” in volume 3.
- “Instruction Prefixes” in volume 3.

F2XM1 Floating-Point Compute 2^x-1

Raises 2 to the power specified by the value in ST(0), subtracts 1, and stores the result in ST(0). The source value must be in the range –1.0 to +1.0. The result is undefined for source values outside this range.

This instruction, when used in conjunction with the FYL2X instruction, can be applied to calculate z = xy by taking advantage of the log property $x^y = 2^{y*\log_2 x}$.

Mnemonic	Opcode	Description
F2XM1	D9 F0	Compute ($2^{ST(0)}$ - 1) and store the result in ST(0).

Related Instructions

FYL2X, FYL2XP1

rFLAGS Affected

None

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	No round up if inexact result bit (5) is 1.
	1	Round up if inexact result bit (5) is 1.
C2	U	
C3	U	
U indicates undefined. M indicates set to 1 or cleared to zero.		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) were set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	The source operand was an SNaN value or unsupported format.
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Denormalized-operand exception (DE)	X	X	X	The result was a denormal operand.
Underflow exception (UE)	X	X	X	The rounded result was too small to fit into the floating-point format of the destination operand.
Precision exception (PE)	X	X	X	The result could not be represented exactly in the destination format.

FABS Floating-Point Absolute Value

Converts the value in ST(0) to its absolute value by clearing the sign bit. The resulting value depends upon the type of number used as the source value:

Source Value (ST(0))	Result (ST(0))
-∞	+∞
-FiniteReal	+FiniteReal
-0	+0
+0	+0
+FiniteReal	+FiniteReal
+∞	+∞
NaN	NaN

This operation applies even if the value in ST(0) is negative zero or negative infinity.

Mnemonic	Opcode	Description
FABS	D9 E1	Replace ST(0) with its absolute value.

Related Instructions

FPREM, FRNDINT, FXTRACT, FCHS

rFLAGS Affected

None

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	If no other flags are set.
C2	U	
C3	U	
U indicates 'undefined.' M indicates set to 1 or cleared to zero.		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) was set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.

FADD FADDP FIADD — Floating-Point Add

Adds two values and stores the result in a floating-point register. If two operands are specified, the values are in ST(0) and another floating-point register and the instruction stores the result in the first register specified. If one operand is specified, the instruction adds the 32-bit or 64-bit floating-point value in the specified memory location to the value in ST(0).

The FADDP instruction adds the value in ST(0) to the value in another floating-point register and pops the register stack. If two operands are specified, the first operand is the other register. If no operand is specified, then the other register is ST(1).

The FIADD instruction reads a 16-bit or 32-bit integer value from the specified memory location, converts it to double-extended-real format, and adds it to the value in ST(0).

Related Instructions

Mnemonic	Opcode	Description
FADD *mem32real*	D8 /0	Replace ST(0) with the sum of ST(0) and the 32-bit real number in memory.
FADD *mem64real*	DC /0	Replace ST(0) with the sum of ST(0) and the 64-bit real number in memory.
FADD ST(0),ST(*i*)	D8 C0+*i*	Replace ST(0) with the sum of ST(0) and ST(*i*).
FADD ST(*i*),ST(0)	DC C0+*i*	Replace ST(*i*) with the sum of ST(0) and ST(*i*).
FADDP	DE C1	Replace ST(1) with the sum of ST(0) and ST(1), and pop the x87 register stack.
FADDP ST(*i*),ST(0)	DE C0+*i*	Replace ST(*i*) with the sum of ST(0) and ST(*i*), and pop the x87 register stack.
FIADD *mem16int*	DE /0	Replace ST(0) with the sum of ST(0) and the 16-bit integer in memory.
FIADD *mem32int*	DA /0	Replace ST(0) with the sum of ST(0) and the 32-bit integer in memory.

FADDP, FIADD

rFLAGS Affected

None

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	No round up, if inexact result bit (5) is 1.
	1	Round up, if inexact result bit (5) is 1.
C2	U	
C3	U	
U indicates 'undefined.' M indicates set to 1 or cleared to zero.		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) was set to 1.
Stack, #SS	X	X	X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking was enabled.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	The source operand was an SNaN value or unsupported format.
	X	X	X	The operands were infinities of unlike sign.
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Denormalized-operand exception (DE)	X	X	X	The result was a denormal operand.
Overflow exception (OE)	X	X	X	The rounded result was too large to fit into the floating-point format of the destination operand.
Underflow exception (UE)	X	X	X	The rounded result was too small to fit into the floating-point format of the destination operand.
Precision exception (PE)	X	X	X	The result could not be represented exactly in the destination format.

FBLD Floating-Point Load Binary-Coded Decimal

Converts a BCD value into double-extended-precision format and pushes the result onto the x87 stack. In the process, it preserves the sign of the source value.

The packed BCD digits should be in the range 0 to 9. Attempting to load invalid digits (Ah through Fh) produces undefined results.

Mnemonic	Opcode	Description
FBLD *mem80dec*	DF /4	Convert a packed BCD value to floating-point and push the result onto the x87 register stack.

Related Instructions

FBSTP

rFLAGS Affected

None

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	If no other flags are set.
C2	U	
C3	U	
U indicates 'undefined.' M indicates set to 1 or cleared to zero.		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) was set to 1.
Stack, #SS	X	X	X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking is enabled.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow occurred.

FBSTP Floating-Point Store Binary-Coded Decimal and Pop

Converts the value in ST(0) to an 18-digit packed BCD integer, stores the result in the specified memory location, and pops the register stack. It rounds a non-integral value to an integer value, depending on the rounding mode specified by the RC field of the x87 control word.

The operand specifies the memory address of the first byte of the resulting 10-byte value.

Mnemonic	Opcode	Description
FBSTP *mem80dec*	DF /6	Convert the floating-point value in ST(0) to BCD, store the result in mem80, and pop the x87 register stack.

Related Instructions

FBLD

rFLAGS Affected

None

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	No round up, if inexact result bit (5) is 1.
	1	Round up, if inexact result bit (5) is 1.
C2	U	
C3	U	
U indicates 'undefined.' M indicates set to 1 or cleared to zero.		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) was set to 1.
Stack, #SS	X	X	X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	Result was located in a nonwritable segment.
			X	A null data segment was used to reference memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking is enabled.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	Source operand was empty or contained a NaN, an unsupported format, or value that exceeded 18 BCD digits in length.
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Inexact result (5) (precision)	X	X	X	The result could not be represented exactly in the destination format.

FCHS Floating-Point Change Sign

Compliments the sign bit of ST(0), changing the value from negative to positive or vice versa. This operation applies to positive and negative floating point values, as well as –0 and +0, NaNs, and +∞ and –∞.

Mnemonic	Opcode	Description
FCHS	D9 E0	Reverse the sign bit of ST(0).

Related Instructions

FABS, FPREM, FRNDINT, FXTRACT

rFLAGS Affected

None

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	If no other flags are set.
C2	U	
C3	U	
U indicates 'undefined.' M indicates set to 1 or cleared to zero.		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) was set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.

FNCLEX (FCLEX) Floating-Point Clear Flags

The FNCLEX instruction clears the following flags in the x87 status word:

- Floating-point exception flags (PE, UE, OE, ZE, DE, and IE)
- Exception summary status flag (ES)
- Stack fault flag (SF)
- Busy flag (B)

It leaves the four condition-code bits undefined. It does not check for possible floating-point exceptions before clearing the flags.

Assemblers usually provide an FCLEX macro that expands into the instruction sequence

```
WAIT                    ; Opcode 9B
FNCLEX destination       ; Opcode DD /7 or DF E0
```

The WAIT (9Bh) instruction checks for pending x87 exceptions and calls an exception handler, if necessary. The FNCLEX instruction then clears all the relevant x87 exception flags.

Mnemonic	Opcode	Description
FNCLEX	DB E2	Clear the floating-point flags without checking for pending unmasked floating-point exceptions.
FCLEX	9B DB E2	Perform a WAIT (9B) to check for pending floating-point exceptions, and then clear the floating-point exception flags.

Related Instructions

WAIT

rFLAGS Affected

None

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	If no other flags are set.
C2	U	
C3	U	
U indicates 'undefined.' M indicates set to 1 or cleared to zero.		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) was set to 1.

FCMOV*cc* Floating-Point Conditional Move

Tests the flags in the rFLAGS register and, depending upon the values encountered, moves the value in another stack register to ST(0).

This set of instructions includes the mnemonics FCMOVB, FCMOVBE, FCMOVE, FCMOVNB, FCMOVNE, FCMOVNU, and FCMOVU.

Use the CPUID instruction to determine if this instruction is supported on a particular x86-64 implementation. It is supported if both the CMOV and FPU bits are set to 1.

Mnemonic	Opcode	Description
FCMOVB ST(0),ST(*i*)	DA C0+*i*	Move the contents of ST(*i*) into ST(0) if the carry flag (CF) is 1.
FCMOVBE ST(0),ST(*i*)	DA D0+*i*	Move the contents of ST(*i*) into ST(0) if the carry flag (CF) is 1 or the zero flag is 1.
FCMOVE ST(0),ST(*i*)	DA C8+*i*	Move the contents of ST(*i*) into ST(0) if the zero flag (ZF) is 1.
FCMOVNB ST(0),ST(*i*)	DB C0+*i*	Move the contents of ST(*i*) into ST(0) if the carry flag (CF) is 0.
FCMOVNBE ST(0),ST(*i*)	DB D0+*i*	Move the contents of ST(*i*) into ST(0) if the carry flag (CF) is 0 and the Zero Flag is 0.
FCMOVNE ST(0),ST(*i*)	DB C8+*i*	Move the contents of ST(*i*) into ST(0) if the zero flag (ZF) is 0.
FCMOVNU ST(0),ST(*i*)	DB D8+*i*	Move the contents of ST(*i*) into ST(0) if the parity flag (PF) is 0.
FCMOVU ST(0),ST(*i*)	DA D8+*i*	Move the contents of ST(*i*) into ST(0) if the parity flag (PF) is 1.

Related Instructions

FCMOVB, FCMOVE, FCMOVBE, FCMOVU, FCMOVNB, FCMOVNE, FCMOVBE, FCNOVNU

rFLAGS Affected

None

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
C2	U	
C3	U	
U indicates undefined. M indicates set to 1 or cleared to zero.		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The Conditional Move instructions are not supported, as indicated by bit 15 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) was set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.

FCOM FCOMP FCOMPP — Floating-Point Compare

Compares the specified value to the value in ST(0) and sets the C0, C2, and C3 condition code flags in the x87 status word as shown in the x87 Condition Code table below. The specified value can be in a floating-point register or a memory location.

The no-operand version compares the value in ST(1) with the value in ST(0).

The comparison operation ignores the sign of zero (–0.0 = +0.0).

After performing the comparison operation, the FCOMP instruction pops the x87 register stack and the FCOMPP instruction pops the x87 register stack twice.

If either or both of the compared values is a NaN or is in an unsupported format, the FCOMx instruction sets the invalid-operation exception (IE) bit in the x87 status word to 1. Then, if the exception is masked (IM bit set to 1 in the x87 control word), the instruction sets the condition flags to "unordered." If the exception is unmasked (IM bit cleared to 0), the instruction does not set the condition code flags.

The FUCOMx instructions perform the same operations as the FCOMx instructions, but do not set the IE bit for QNaNs.

Mnemonic	Opcode	Description
FCOM	D8 D1	Compare the contents of ST(0) to the contents of ST(1) and set condition flags to reflect the results of the comparison.
FCOM *mem32real*	D8 /2	Compare the contents of *mem32real* to the contents of ST(0) and set condition flags to reflect the results of the comparison.
FCOM *mem64real*	DC /2	Compare the contents of *mem64real* to the contents of ST(0) and set condition flags to reflect the results of the comparison.
FCOM ST(*i*)	D8 D0+*i*	Compare the contents of ST(*i*) to the contents of ST(0) and set condition flags to reflect the results of the comparison.
FCOMP	D8 D9	Compare the contents of ST(0) to the contents of ST(1), set condition flags to reflect the results of the comparison, and pop the x87 register stack.
FCOMP *mem32real*	D8 /3	Compare the contents of *mem32real* to the contents of ST(0), set condition flags to reflect the results of the comparison, and pop the x87 register stack.

FCOMP *mem64real*	DC /3	Compare the contents of *mem64real* to the contents of ST(0), set condition flags to reflect the results of the comparison, and pop the x87 register stack.
FCOMP ST(*i*)	D8 D8+*i*	Compare the contents of ST(*i*) to the contents of ST(0), set condition flags to reflect the results of the comparison, and pop the x87 register stack.
FCOMPP	DE D9	Compare the contents of ST(0) to the contents of ST(1), set condition flags to reflect the results of the comparison, and pop the x87 register stack twice.

Related Instructions

FCOM, FCOMPP, FCOMI, FCOMIP, FICOM, FICOMP, FTST, FUCOMI, FUCOMIP, FXAM

rFLAGS Affected

None

x87 Condition Code

C3	C2	C0	Description
0	0	0	ST(0) > source
0	0	1	ST(0) < source
1	0	0	ST(0) = source
1	1	1	Unordered (numbers cannot be compared)

x87 Condition Code	Value	Description
C0	0/1	See previous table for interpretation.
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	If no other flags are set.
C2	0/1	See previous table for interpretation.
C3	0/1	See previous table for interpretation.

U indicates 'undefined.' M indicates set to 1 or cleared to zero.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) was set to 1.
Stack, #SS	X	X	X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking is enabled.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	One or both operands were NaN values or were in an unsupported format.
	X	X	X	Register was marked empty.
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Denormalized-operand exception (DE)	X	X	X	One or both operands were denormal values.

FCOMI FCOMIP — Floating-Point Compare and Set Flags

Compares the value in ST(0) with the value in another floating-point register and sets the zero flag (ZF), parity flag (PF), and carry flag (CF) in the rFLAGS register based on the result as shown in the table in the x87 Condition Code section.

The comparison operation ignores the sign of zero (–0.0 = +0.0).

After performing the comparison operation, FCOMIP pops the x87 register stack.

If either or both of the compared values is a NaN or is in an unsupported format, the FCOMIx instruction sets the invalid-operation exception (IE) bit in the x87 status word to 1. Then, if the exception is masked (IM bit set to 1 in the x87 control word), the instruction sets the flags to "unordered." If the exception is unmasked (IM bit cleared to 0), the instruction does not set the flags.

The FUCOMIx instructions perform the same operations as the FCOMIx instructions, but do not set the IE bit for QNaNs.

Mnemonic	**Opcode**	**Description**
FCOMI ST(0),ST(*i*)	DB F0+*i*	Compare the contents of ST(0) with the contents of ST(*i*) and set status flags to reflect the results of the comparison.
FCOMIP ST(0),ST(*i*)	DF F0+*i*	Compare the contents of ST(0) with the contents of ST(*i*), set status flags to reflect the results of the comparison, and pop the x87 register stack.

Related Instructions

FCOM, FCOMPP, FCOMI, FCOMIP, FICOM, FICOMP, FTST, FUCOMI, FUCOMIP, FXAM

rFLAGS Affected

Flag	ST(0) > ST(i)	ST(0) < ST(i)	ST(0) = ST(i)	Unordered
ZF	0	0	1	1
PF	0	0	0	1
CF	0	1	0	1

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	1	Stack underflow
	0	Otherwise
C2	U	
C3	U	
U indicates 'undefined.' M indicates set to 1 or cleared to zero.		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) was set to 1.
Stack, #SS	X	X	X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking is enabled.

Exception	Real	Virtual 8086	Protected	Cause of Exception
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	One or both operands were NaN values or were in an unsupported format.
	X	X	X	Register was marked empty.
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Denormalized-operand exception (DE)	X	X	X	One or both operands were denormal values.

FCOS Floating-Point Cosine

Computes the cosine of the radian value in ST(0) and stores the result in ST(0).

If the radian value lies outside the valid range of -2^{63} to $+2^{63}$ radians, the instruction sets the C2 flag in the x87 status word to 1 to indicate the value is out of range and does not change the value in ST(0). It does not set any of the exception flags. The program should check the C2 flag and, if necessary, can reduce an invalid source value to the proper range by using the FPREM instruction with the value 2π in ST(1) and the out-of-range radian value in ST(0).

Mnemonic	Opcode	Description
FCOS	D9 FF	Replace ST(0) with the cosine of ST(0).

Related Instructions

FPTAN, FPATAN, FSIN, FSINCOS

rFLAGS Affected

None

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Numeric underflow is set.
	0	No round up, if inexact result bit is 1.
	1	Round up, if inexact result bit is 1.
C2	U	
C3	U	
U indicates 'undefined.' M indicates set to 1 or cleared to zero.		

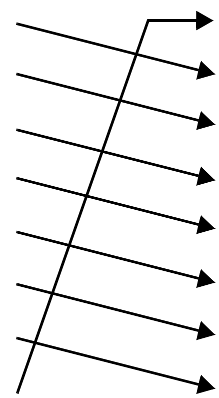
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	The source operand was an SNaN value or unsupported format.
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Denormalized-operand exception (DE)	X	X	X	The result was a denormal operand.
Underflow exception (UE)	X	X	X	The rounded result was too small to fit into the floating-point format of the destination operand.
Precision exception (PE)	X	X	X	The result could not be represented exactly in the destination format.

FDECSTP — Floating-Point Decrement Stack-Top Pointer

Decrements the top-of-stack pointer (TOP) field of the x87 status word. If the TOP field contains 0, it is set to 7. In other words, this instruction rotates the stack by one position.

Mnemonic	Opcode	Description
FDECSTP	D9 F6	Decrement the TOP field in the x87 status word.

Data Register	Before FDECSTP			After FDECSTP	
	Value	Stack Pointer		Stack Pointer	Value
7	num1	ST(7)		ST(0)	num1
6	num2	ST(6)		ST(7)	num2
5	num3	ST(5)		ST(6)	num3
4	num4	ST(4)		ST(5)	num4
3	num5	ST(3)		ST(4)	num5
2	num6	ST(2)		ST(3)	num6
1	num7	ST(1)		ST(2)	num7
0	num8	ST(0)		ST(1)	num8

Related Instructions

FINCSTP

rFLAGS Affected

None

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	If no other flags are set.
C2	U	
C3	U	
U indicates 'undefined.' M indicates set to 1 or cleared to zero.		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) was set to 1.

FDIV FDIVP FIDIV — Floating-Point Divide

Divides the value in a floating-point register by the value in another register or a memory location and stores the result in the register containing the dividend. For the FDIV and FDIVP instructions, the divisor value in memory can be stored in single-precision or double-precision floating-point format.

If only one operand is specified, the instruction divides the value in ST(0) by the value in the specified memory location.

If no operands are specified, the FDIVP instruction divides the value in ST(1) by the value in ST(0), stores the result in ST(1), and pops the x87 register stack.

The FIDIV instruction converts a divisor in word integer or short integer format to double-extended-precision floating-point format before performing the division. It treats an integer 0 as +0.

If the zero-divide exception is not masked (ZM bit cleared to 0 in the x87 control word) and the operation causes a zero-divide exception (sets the ZE bit in the x87 status word to 1), the operation stores no result. If the zero-divide exception is masked (ZM bit set to 1), a zero-divide exception causes ±∞ to be stored.

The sign of the operands, even if one of the operands is 0, determines the sign of the result.

Mnemonic	Opcode	Description
FDIV *mem32real*	D8 /6	Divide ST(0) by the contents of *mem32real* and store the result in ST(0).
FDIV *mem64real*	DC /6	Divide ST(0) by the contents of *mem64real* and store the result in ST(0).
FDIV ST(0),ST(*i*)	D8 F0+*i*	Divide ST(0) by the contents of ST(*i*) and store the result in ST(*i*).
FDIV ST(*i*),ST(0)	DC F8+*i*	Divide ST(*i*) by the contents of ST(0) and store the result in ST(*i*).
FDIVP	DE F9	Divide ST(1) by ST(0), store the result in ST(1), and pop the x87 register stack.

Mnemonic	Opcode	Description
FDIVP ST(*i*),ST(0)	DE F8+*i*	Divide ST(*i*) by ST(0), store the result in ST(*i*), and pop the x87 register stack.
FIDIV *mem32int*	DA /6	Divide ST(0) by the contents of *mem32int* and store the result in ST(0).

Related Instructions

FDIVP, FIDIV, FDIVR, FDIVRP, FIDIVR

rFLAGS Affected

None

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	No round up, if inexact result bit (5) is 1.
	1	Round up, if inexact result bit (5) is 1.
C2	U	
C3	U	
U indicates 'undefined.' M indicates set to 1 or cleared to zero.		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) was set to 1.
Stack, #SS	X	X	X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeds a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking is enabled.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	The source operand was an SNaN value or unsupported format.
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Denormalized-operand exception (DE)	X	X	X	The result was a denormal operand.
Zero-divide exception (ZE)	X	X	X	DEST/±0, where DEST was not equal to ±0.
Overflow exception (OE)	X	X	X	The rounded result was too large to fit into the floating-point format of the destination operand.
Underflow exception (UE)	X	X	X	The rounded result was too small to fit into the floating-point format of the destination operand.
Precision exception (PE)	X	X	X	The result could not be represented exactly in the destination format.

FDIVR FDIVRP FIDIVR — Floating-Point Divide Reverse

Divides a value in a floating-point register or a memory location by the value in a floating-point register and stores the result in the register containing the divisor. For the FDIVR and FDIVRP instructions, a dividend value in memory can be stored in single-precision or double-precision floating-point format.

If one operand is specified, the instruction divides the value at the specified memory location by the value in ST(0). If two operands are specified, it divides the value in ST(0) by the value in another x87 stack register or vice versa.

The FIDIVR instruction converts a dividend in word integer or short integer format to double-extended-precision format before performing the division.

The FDIVRP instruction pops the x87 register stack after performing the division operation. If no operand is specified, the FDIVRP instruction divides the value in ST(0) by the value in ST(1).

If the zero-divide exception is not masked (ZM bit cleared to 0 in the x87 control word) and the operation causes a zero-divide exception (sets the ZE bit in the x87 status word to 1), the operation stores no result. If the zero-divide exception is masked (ZM bit set to 1), a zero-divide exception causes ±∞ to be stored.

The sign of the operands, even if one of the operands is 0, determines the sign of the result.

Mnemonic	Opcode	Description
FDIVR *mem32real*	D8 /7	Divide the contents of *mem32real* by the contents of ST(0) and store the result in ST(0).
FDIVR *mem64real*	DC /7	Divide the contents of *mem64real* by the contents of ST(0) and store the result in ST(0).
FDIVR ST(0),ST(*i*)	D8 F8+*i*	Divide the contents of ST(*i*) by the contents of ST(0) and store the result in ST(0).
FDIVR ST(*i*), ST(0)	DC F0+*i*	Divide the contents of ST(0) by the contents of ST(*i*) and store the result in ST(*i*).
FDIVRP	DE F1	Divide the contents of ST(0) by the contents of ST(1), store the result in ST(1), and pop the x87 register stack.

FDIVRP ST(*i*), ST(0)	DE F0 +*i*	Divide the contents of ST(0) by the contents of ST(*i*), store the result in ST(*i*), and pop the x87 register stack.
FIDIVR *mem16int*	DE /7	Divide the contents of *mem16int* by the contents of ST(0) and store result in ST(0).
FIDIVR *mem32int*	DA /7	Divide the contents of *mem32int* by the contents of ST(0) and store result in ST(0).

Related Instructions

FDIV, FDIVRP, FIDIV, FIDIVR

rFLAGS Affected

None

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	No round up, if inexact result bit (5) is 1.
	1	Round up, if inexact result bit (5) is 1.
C2	U	
C3	U	
U indicates 'undefined.' M indicates set to 1 or cleared to zero.		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) was set to 1.
Stack, #SS	X	X	X	A memory address exceeded the stack segment limit or is non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or is non-canonical.
			X	A null data segment was used to reference memory.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking is enabled.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	The source operand was an SNaN value or unsupported format.
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Denormalized-operand exception (DE)	X	X	X	The result was a denormal operand.
Zero-divide exception (ZE)	X	X	X	SRC /±0, where SRC was not equal to ±0.
Overflow exception (OE)	X	X	X	The rounded result was too large to fit into the floating-point format of the destination operand.
Underflow exception (UE)	X	X	X	The rounded result was too small to fit into the floating-point format of the destination operand.
Precision exception (PE)	X	X	X	The result could not be represented exactly in the destination format.

FFREE — Floating-Point Free Register

Frees the specified x87 stack register by marking its tag register entry as empty. The instruction does not affect the contents of the freed register or the top-of-stack pointer (TOP).

Mnemonic	Opcode	Description
FFREE ST(*i*)	DD C0+*i*	Set the tag for x87 stack register *i* to empty (11b).

Related Instructions

FLD, FST, FSTP

rFLAGS Affected

None

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Numeric underflow.
	0	No round up, if inexact result bit is 1.
	1	Round up, if inexact result bit is 1.
C2	U	
C3	U	
U indicates 'undefined.' M indicates set to 1 or cleared to zero.		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.

FICOM FICOMP — Floating-Point Integer Compare

Converts a 16-bit or 32-bit integer value to double-extended-precision format, compares it to the value in ST(0), and sets the C0, C2, and C3 condition code flags in the x87 status word to reflect the results.

The comparison operation ignores the sign of zero (–0.0 = +0.0).

After performing the comparison operation, the FICOMP instruction pops the x87 register stack.

If either or both of the compared values is a NaN or is in an unsupported format, the instruction sets the condition flags to "unordered."

Mnemonic	Opcode	Description
FICOM *mem16int*	DE /2	Convert the contents of *mem16int* to double-extended-precision format, compare the result to the contents of ST(0), and set condition flags to reflect the results of the comparison.
FICOM *mem32int*	DA /2	Convert the contents of *mem32int* to double-extended-precision format, compare the result to the contents of ST(0), and set condition flags to reflect the results of the comparison.
FICOMP *mem16int*	DE /3	Convert the contents of *mem16int* to double-extended-precision format, compare the result to the contents of ST(0), set condition flags to reflect the results of the comparison, and pop the x87 register stack.
FICOMP *mem32int*	DA /3	Convert the contents of *mem32int* to double-extended-precision format, compare the result to the contents of ST(0), set condition flags to reflect the results of the comparison, and pop the x87 register stack.

Related Instructions

FCOM, FCOMPP, FCOMI, FCOMIP, FICOM, FICOMP, FTST, FUCOMI, FUCOMIP, FXAM

rFLAGS Affected

None

x87 Condition Code

C3	C2	C0	Description
0	0	0	ST(0) > source
0	0	1	ST(0) < source
1	0	0	ST(0) = source
1	1	1	numbers cannot be compared

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) was set to 1.
Stack, #SS	X	X	X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking is enabled.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	One or both operands were NaN values or unsupported formats.
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Denormalized-operand exception (DE)	X	X	X	One or both operands were denormal values.

FILD Floating-Point Load Integer

Converts a signed-integer value in memory to double-extended-precision format and pushes the value onto the x87 register stack without rounding errors, preserving the sign of the value. The value can be a 16-bit, 32-bit, or 64- bit integer value.

Mnemonic	Opcode	Description
FILD *mem16int*	DF /0	Push the contents of *mem16int* onto the x87 register stack.
FILD *mem32int*	DB /0	Push the contents of *mem32int* onto the x87 register stack.
FILD *mem64int*	DF /5	Push the contents of *mem64int* onto the x87 register stack.

Related Instructions

FLD, FST, FSTP, FIST, FISTP, FBLD, FBSTP

rFLAGS Affected

None

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	If no other flags are set.
C2	U	
C3	U	
U indicates 'undefined.' M indicates set to 1 or cleared to zero.		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
Stack, #SS	X	X	X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking is enabled.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow occurred.

FINCSTP Floating-Point Increment Stack-Top Pointer

Increments the top-of-stack pointer (TOP) field of the x87 status word. If the TOP field contains 7, it is set to 0. In other words, this instruction rotates the stack by one position.

Mnemonic	Opcode	Description
FINCSTP	D9 F7	Increment the TOP field in the x87 status word.

Data Register	Before FINCSTP			After FINCSTP	
	Value	Stack Pointer		Stack Pointer	Value
7	num1	ST(7)		ST(6)	num1
6	num2	ST(6)		ST(5)	num2
5	num3	ST(5)		ST(4)	num3
4	num4	ST(4)		ST(3)	num4
3	num5	ST(3)		ST(2)	num5
2	num6	ST(2)		ST(1)	num6
1	num7	ST(1)		ST(0)	num7
0	num8	ST(0)		ST(7)	num8

Related Instructions

FDECSTP

rFLAGS Affected

None

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	If no other flags are set.
C2	U	
C3	U	
U indicates 'undefined.' M indicates set to 1 or cleared to zero.		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) was set to 1.

FNINIT (FINIT) Floating-Point Initialize

Sets the x87 control word register, status word register, tag word register, instruction pointer, and data pointer to their default states as follows:

- Sets the x87 control word to 037Fh—round to nearest (RC = 00b); double-extended-precision (PC = 11b); all exceptions masked (PM, UM, OM, ZM, DM, and IM all set to 1).
- Clears all bits in the x87 status word (TOP is set to 0, which maps ST(0) onto FPR0).
- Marks all x87 stack registers as empty (11b) in the x87 tag register.
- Clears the instruction pointer and the data pointer.

These instructions do not change any values the x87 stack registers.

Assemblers usually provide an FINIT macro that expands into the instruction sequence

```
WAIT                    ; Opcode 9B
FNINIT destination      ; Opcode DB E3
```

The WAIT (9Bh) instruction checks for pending x87 exceptions and calls an exception handler, if necessary. The FNINIT instruction then resets all the relevant x87 registers to their default states.

Mnemonic	Opcode	Description
FNINIT	DB E3	Initialize the x87 unit without checking for unmasked floating-point exceptions.
FINIT	9B DB E3	Perform a WAIT (9B) to check for pending floating-point exceptions and then initialize the x87 unit.

Related Instructions

FWAIT, WAIT

rFLAGS Affected

None

x87 Condition Code

x87 Condition Code	Value	Description
C0	0	
C1	0	
C2	0	
C3	0	
U indicates 'undefined.' M indicates set to 1 or cleared to zero.		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) was set to 1.

FIST FISTP Floating-Point Integer Store

Converts the value in ST(0) to a signed integer, rounds it if necessary, and copies it to the specified memory location. The rounding control (RC) field of the x87 control word determines the type of rounding used. The default is round-to-nearest (00b).

The FIST instruction supports 16-bit and 32-bit values. The FISTP instructions supports 16-bit, 32-bit, and 64-bit values.

The FISTP instruction pops the stack after storing the rounded value in memory.

If the value is too large for the destination location, is a NaN, or is in an unsupported format, the instruction sets the invalid-operation exception (IE) bit in the x87 status word to 1. Then, if the exception is masked (IM bit set to 1 in the x87 control word), the instruction stores the integer indefinite value. If the exception is unmasked (IM bit cleared to 0), the instruction does not store the value.

Mnemonic	Opcode	Description
FIST *mem16int*	DF /2	Convert the contents of ST(0) to integer and store the result in *mem16int*.
FIST *mem32int*	DB /2	Convert the contents of ST(0) to integer and store the result in *mem32int*.
FISTP *mem16int*	DF /3	Convert the contents of ST(0) to integer, store the result in *mem16int*, and pop the x87 register stack.
FISTP *mem32int*	DB /3	Convert the contents of ST(0) to integer, store the result in *mem32int*, and pop the x87 register stack.
FISTP *mem64int*	DF /7	Convert the contents of ST(0) to integer, store the result in *mem64int*, and pop the x87 register stack.

The following table shows the results of storing various types of numbers as integers.

ST(0)	DEST
-∞	Invalid-operation (IE) exception
–Finite-real < –1	–Integer
–1 < –Finite-real< –0	0 or ±1, depending on the rounding mode
-0	0
+0	0

ST(0)	DEST
+0 < +Finite-real < +1	0 or ±1, depending on the rounding mode
+Finite-real > +1	+Integer
+∞	Invalid-operation (IE) exception
NaN	Invalid-operation (IE) exception

Related Instructions

FLD, FST, FSTP, FILD, FISTP, FBLD, FBSTP

rFLAGS Affected

None

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	No round up, if inexact result bit (5) is 1.
	1	Round up, if inexact result bit (5) is 1.
C2	U	
C3	U	

U indicates 'undefined.' M indicates set to 1 or cleared to zero.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) was set to 1.
Stack, #SS	X	X	X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	Result is located in a nonwritable segment.
			X	A null data segment is being used to reference memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking is enabled.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	The source operand was too large for the destination format.
	X	X	X	The source operand was a NaN value or unsupported format.
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Precision exception (PE)	X	X	X	The result could not be represented exactly in the destination format.

FLD Floating-Point Load

Pushes a value in memory or in a floating-point register onto the register stack. If in memory, the value can be a single-precision, double-precision, or double-extended-precision floating-point value. The operation converts a single-precision or double-precision value to double-extended-precision format before pushing it onto the stack.

Mnemonic	Opcode	Description
FLD *mem32real*	D9 /0	Push the contents of *mem32real* onto the x87 register stack.
FLD *mem64real*	DD /0	Push the contents of *mem64real* onto the x87 register stack.
FLD *mem80real*	DB /5	Push the contents of *mem80real* onto the x87 register stack.
FLD ST(*i*)	D9 C0+*i*	Push the contents of ST(*i*) onto the x87 register stack.

Related Instructions

FFREE, FLD, FST, FSTP, FILD, FIST, FISTP, FBLD, FBSTP

rFLAGS Affected

None

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	If no other flags are set.
C2	U	
C3	U	
U indicates 'undefined.' M indicates set to 1 or cleared to zero.		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) was set to 1.
Stack, #SS	X	X	X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	Result was located in a nonwritable segment.
			X	A null data segment was used to reference memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking is enabled.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	The source operand was an SNaN value or unsupported format.
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow occurred.
Denormalized-operand exception (DE)	X	X	X	The source operand was a denormal value. This exception does not occur if the source operand was in double-extended-precision format.

FLD1 Floating-Point Load +1.0

Pushes the floating-point value +1.0 onto the register stack.

Mnemonic	Opcode	Description
FLD1	D9 E8	Push +1.0 onto the x87 register stack.

Related Instructions

FLD, FLDZ, FLDPI, FLDL2T, FLDL2E, FLDLG2, FLDLN2

rFLAGS Affected

None

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	If no other flags are set.
C2	U	
C3	U	
U indicates 'undefined.' M indicates set to 1 or cleared to zero.		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow occurred.

FLDCW Floating-Point Load x87 Control Word

Loads a 16-bit value from the specified memory location into the x87 control word. If the new x87 control word unmasks any pending floating point exceptions, then they are handled upon execution of the next x87 floating-point or 64-bit media instruction.

To avoid generating exceptions when loading a new control word, use the FCLEX or FNCLEX instruction to clear any pending exceptions.

Mnemonic	Opcode	Description
FLDCW *mem2env*	D9 /5	Load the contents of *mem2env* into the x87 control word.

Related Instructions

FSTCW, FNSTCW, FSTSW, FNSTSW, FSTENV, FNSTENV, FLDENV, FCLEX, FNCLEX

rFLAGS Affected

None

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Numeric underflow is set.
	0	No round up, if inexact result bit is 1.
	1	Round up, if inexact result bit is 1.
C2	U	
C3	U	
U indicates 'undefined.' M indicates set to 1 or cleared to zero.		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) was set to 1.
Stack, #SS	X	X	X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking is enabled.

FLDENV Floating-Point Load x87 Environment

Restores the x87 environment from memory starting at the specified address. The x87 environment consists of the x87 control, status, and tag word registers, the last non-control x87 instruction pointer, the last x87 data pointer, and the opcode of the last completed non-control x87 instruction.

The x87 environment requires a 14-byte or 28-byte area in memory, depending on whether the processor is operating in protected or real mode and whether the operand-size attribute is 16-bit or 32-bit. See "Media and x87 Processor State" in volume 2 for details on how this instruction stores the x87 environment in memory.

The environment to be loaded is typically stored by a previous FNSTENV or FSTENV instruction. The FLDENV instruction should be executed in the same operating mode as the instruction that stored the x87 environment.

If FLDENV results in set exception flags in the loaded x87 status word register, and these exceptions are unmasked in the x87 control word register, a floating-point exception occurs when the next floating-point instruction is executed (except for the no-wait floating-point instructions).

To avoid generating exceptions when loading a new environment, use the FCLEX or FNCLEX instruction to clear the exception flags in the x87 status word before storing that environment.

Mnemonic	Opcode	Description
FLDENV *mem14/28env*	D9 /4	Load the complete contents of the x87 environment from *mem14/28env*.

Related Instructions

FSTENV, FNSTENV, FCLEX, FNCLEX

rFLAGS Affected

None

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Numeric underflow is set.
	0	No round up, if inexact result bit is 1.
	1	Round up, if inexact result bit is 1.
C2	U	
C3	U	
U indicates 'undefined.' M indicates set to 1 or cleared to zero.		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) was set to 1.
Stack, #SS	X	X	X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking is enabled.

FLDL2E Floating-Point Load Log_2 e

Pushes log_2e onto the x87 register stack. The value in ST(0) is the result, in double-extended-precision format, of rounding an internal 66-bit constant according to the setting of the RC field in the x87 control word register.

Mnemonic	Opcode	Description
FLDL2E	D9 EA	Push $\log_2 e$ onto the x87 register stack.

Related Instructions

FLD, FLD1, FLDZ, FLDPI, FLDL2T, FLDLG2, FLDLN2

rFLAGS Affected

None

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	If no other flags are set.
C2	U	
C3	U	
U indicates 'undefined.' M indicates set to 1 or cleared to zero.		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) was set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow occurred.

FLDL2T Floating-Point Load $\log_2 10$

Pushes $\log_2 10$ onto the x87 register stack. The value in ST(0) is the result, in double-extended-precision format, of rounding an internal 66-bit constant according to the setting of the RC field in the x87 control word register.

Mnemonic	Opcode	Description
FLDL2T	D9 E9	Push $\log_2 10$ onto the x87 register stack.

Related Instructions

FLD, FLD1, FLDZ, FLDPI, FLDL2E, FLDLG2, FLDLN2

rFLAGS Affected

None

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	If no other flags are set.
C2	U	
C3	U	
U indicates 'undefined.' M indicates set to 1 or cleared to zero.		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) was set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow occurred.

FLDLG2 Floating-Point Load Log_{10} 2

Pushes $\log_{10} 2$ onto the x87 register stack. The value in ST(0) is the result, in double-extended-precision format, of rounding an internal 66-bit constant according to the setting of the RC field in the x87 control word register.

Mnemonic	Opcode	Description
FLDLG2	D9 EC	Push $\log_{10}2$ onto the x87 register stack.

Related Instructions

FLD, FLD1, FLDZ, FLDPI, FLDL2T, FLDL2E, FLDLN2

rFLAGS Affected

None

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	If no other flags are set.
C2	U	
C3	U	
U indicates 'undefined.' M indicates set to 1 or cleared to zero.		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) was set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow occurred.

FLDLN2 Floating-Point Load Ln 2

Pushes $\log_e 2$ onto the x87 register stack. The value in ST(0) is the result, in double-extended-precision format, of rounding an internal 66-bit constant according to the setting of the RC field in the x87 control word register.

Mnemonic	Opcode	Description
FLDLN2	D9 ED	Push $\log_e 2$ onto the x87 register stack.

Related Instructions

FLD, FLD1, FLDZ, FLDPI, FLDL2T, FLDL2E, FLDLG2

rFLAGS Affected

None

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	If no other flags are set.
C2	U	
C3	U	

U indicates 'undefined.' M indicates set to 1 or cleared to zero.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) was set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow occurred.

FLDPI Floating-Point Load Pi

Pushes π onto the x87 register stack. The value in ST(0) is the result, in double-extended-precision format, of rounding an internal 66-bit constant according to the setting of the RC field in the x87 control word register.

Mnemonic	Opcode	Description
FLDPI	D9 EB	Push π onto the x87 register stack.

Related Instructions

FLD, FLD1, FLDZ, FLDL2T, FLDL2E, FLDLG2, FLDLN2

rFLAGS Affected

None

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	If no other flags are set.
C2	U	
C3	U	
U indicates 'undefined.' M indicates set to 1 or cleared to zero.		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) was set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow occurred.

FLDZ Floating-Point Load +0.0

Pushes +0.0 onto the x87 register stack.

Mnemonic	Opcode	Description
FLDZ	D9 EE	Push zero onto the x87 register stack.

Related Instructions

FLD, FLD1, FLDPI, FLDL2T, FLDL2E, FLDLG2, FLDLN2

rFLAGS Affected

None

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	If no other flags are set.
C2	U	
C3	U	
U indicates 'undefined.' M indicates set to 1 or cleared to zero.		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) was set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow occurred.

FMUL FMULP FIMUL — Floating-Point Multiply

Multiplies the value in a floating-point register by the value in a memory location or another stack register and stores the result in the first register. The instruction converts a single-precision or double-precision value in memory to double-extended-precision format before multiplying.

If one operand is specified, the instruction multiplies the value in the ST(0) register by the value in the specified memory location and stores the result in the ST(0) register.

If two operands are specified, the instruction multiplies the value in the ST(0) register by the value in another specified floating-point register and stores the result in the register specified in the first operand.

The FMULP instruction pops the x87 stack after storing the product. The no-operand version of the FMULP instruction multiplies the value in the ST(1) register by the value in the ST(0) register and stores the product in the ST(1) register.

The FIMUL instruction converts an short-integer or word-integer value in memory to double-extended-precision format, multiplies it by the value in ST(0), and stores the product in ST(0).

Mnemonic	Opcode	Description
FMUL *mem32real*	D8 /1	Multiply the contents of *mem32real* by the contents of ST(0) and store the result in ST(0).
FMUL *mem64real*	DC /1	Multiply the contents of *mem64real* by the contents of ST(0) and store the result in ST(0).
FMUL ST(0),ST(*i*)	D8 C8+*i*	Multiply ST(0) by ST(*i*) and store the result in ST(0).
FMUL ST(*i*),ST(0)	DC C8+*i*	Multiply ST(0) by ST(*i*) and store the result in ST(*i*).
FMULP	DE C9	Multiply the contents of ST(1) by the contents of ST(0), store the result in ST(1), and pop the x87 register stack.
FMULP ST(*i*),ST(0)	DE C8+*i*	Multiply ST(0) by ST(*i*), store the result in ST(*i*), and pop the x87 register stack.

FIMUL *mem16int*	DE /1	Multiply the contents of *mem16int* by the contents of ST(0) and store the result in ST(0).
FIMUL *mem32int*	DA /1	Multiply the contents of *mem32int* by the contents of ST(0) and store the result in ST(0).

Related Instructions

FMULP, FIMUL

rFLAGS Affected

None

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	No round up, if inexact result bit (5) is 1.
	1	Round up, if inexact result bit (5) is 1.
C2	U	
C3	U	
U indicates 'undefined.' M indicates set to 1 or cleared to zero.		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) was set to 1.
Stack, #SS	X	X	X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking is enabled.

Exception	Real	Virtual 8086	Protected	Cause of Exception
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	The source operand was an SNaN value or unsupported format.
	X	X	X	One operand was ±0 and the other was ±infinity.
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Denormalized-operand exception (DE)	X	X	X	The source operand was a denormal value.
Overflow exception (OE)	X	X	X	The rounded result was too large to fit into the floating-point format of the destination operand.
Underflow exception (UE)	X	X	X	The rounded result was too small to fit into the floating-point format of the destination operand.
Inexact result (5) (precision)	X	X	X	The result could not be represented exactly in the destination format.

FNOP Floating-Point No Operation

Performs no operation. This instruction affects only the rIP register. It does not otherwise affect the processor context.

Mnemonic	Opcode	Description
FNOP	D9 D0	Perform no operation.

Related Instructions

FWAIT, NOP

rFLAGS Affected

None

x87 Condition Code

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) was set to 1.

FNSAVE (FSAVE) Floating-Point Save No-Wait x87 and MMX™ State

Stores the complete x87 state to memory starting at the specified address and reinitializes the x87 state. The x87 state requires 94 or 108 bytes of memory, depending upon whether the processor is operating in real or protected mode and whether the operand-size attribute is 16-bit or 32-bit. Because the MMX registers are mapped onto the low 64 bits of the x87 floating-point registers, this operation also saves the MMX state. For details about the memory image saved by FNSAVE, see "Media and x87 Processor State" in volume 2.

The FNSAVE instruction does not wait for pending unmasked x87 floating-point exceptions to be processed. Processor interrupts should be disabled before using this instruction.

Assemblers usually provide an FSAVE macro that expands into the instruction sequence

```
WAIT                    ; Opcode 9B
FNSAVE destination      ; Opcode DD /6
```

The WAIT (9Bh) instruction checks for pending x87 exceptions and calls an exception handler, if necessary. The FNSAVE instruction then stores the x87 state to the specified destination.

Mnemonic	Opcode	Description
FNSAVE *mem94/108env*	DD /6	Copy the x87 state to *mem94/108env* without checking for pending floating-point exceptions, then reinitialize the x87 state.
FSAVE *mem94/108env*	9B DD /6	Copy the x87 state to *mem94/108env* after checking for pending floating-point exceptions, then reinitialize the x87 state.

Related Instructions

FSAVE, FRSTOR, FXSAVE, FXRSTOR

rFLAGS Affected

None

x87 Condition Code

x87 Condition Code	Description
C0	0
C1	0
C2	0
C3	0

Exceptions (All Modes)

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) was set to 1.
Stack, #SS	X	X	X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	Result was located in a nonwritable segment.
			X	A null data segment was used to reference memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking is enabled.

FPATAN Floating-Point Partial Arctangent

Computes the arctangent of the ordinate (Y) in ST(1) divided by the abscissa (X) in ST(0), which is the angle in radians between the X axis and the radius vector from the origin to the point (X, Y). It then stores the result in ST(1) and pops the x87 register stack. The resulting value has the same sign as the ordinate value and a magnitude less than or equal to π.

There is no restriction on the range of values that FPATAN can accept. The following table shows the results obtained when computing the arctangent of various classes of numbers, assuming that underflow does not occur:

		X (ST(0))						
		−∞	**−F[a]**	**−0**	**+0**	**+F**	**+∞**	**NaN**
Y (ST(1))	**−∞**	−3π/4	−π/2	−π/2	−π/2	−π/2	−π/4	NaN
	−F	−π	−π to −π/2	−π/2	−π/2	−π/2 to −0	−0	NaN
	−0	−π	−π	−π	−0	−0	−0	NaN
	+0	+π	+π	+π	+0	+0	+0	NaN
	+F	+π	+π to +π/2	+π/2	+π/2	+π/2 to +0	+0	NaN
	+∞	+3π/4	+π/2	+π/2	+π/2	+π/2	+π/4	NaN
	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

a.F means finite-real number.

Mnemonic	Opcode	Description
FPATAN	D9 F3	Compute arctan(ST(1)/ST(0)), store the result in ST(1), and pop the x87 register stack.

Related Instructions

FCOS, FPATAN, FSIN, FSINCOS

rFLAGS Affected

None

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	No round up, if inexact result bit (5) is 1.
	1	Round up, if inexact result bit (5) is 1.
C2	U	
C3	U	
U indicates 'undefined.' M indicates set to 1 or cleared to zero.		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	The source operand was an SNaN value or unsupported format.
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Denormalized-operand exception (DE)	X	X	X	The source operand was a denormal value.
Underflow exception (UE)	X	X	X	The rounded result was too small to fit into the floating-point format of the destination operand.
Inexact result (5) (precision)	X	X	X	The result could not be represented exactly in the destination format.

FPREM Floating-Point Partial Remainder

Computes the exact remainder obtained by dividing the value in ST(0) by that in ST(1), and stores the result in ST(0). It computes the remainder by an iterative subtract-and-shift long division algorithm in which one quotient bit is calculated in each iteration.

If the exponent difference between ST(0) and ST(1) is less than 64, the instruction computes all integer bits of the quotient, guaranteeing that the remainder is less in magnitude than the divisor in ST(1). If the exponent difference is equal to or greater than 64, it computes only the subset of integer quotient bits numbering between 32 and 63, returns a partial remainder, and sets the C2 condition code bit to 1.

FPREM is supported for software that was written for early x87 coprocessors. Unlike the FPREM1 instruction, FPREM does not compute the partial remainder as specified in IEEE Standard 754.

Mnemonic	Opcode	Description
FPREM	D9 F8	Compute the remainder of the division of ST(0) by ST(1) and store the result in ST(0).

```
ExpDiff = Exponent(ST(0)) - Exponent(ST(1);
if (ExpDiff < 0){
   SW.C2 = 0;
   {SW.C0, SW.C3, SW.C1} = 0;
}
else if (ExpDiff < 64){
   Quotient = Floor(ST(0)/ST(1));
   ST(0) = ST(0) - (ST(1) * Quotient);
   SW.C2 = 0;
   {SW.C0, SW.C3, SW.C1} = Quotient mod 8;
}
else{
   N = 32 + (ExpDiff mod 32);
   Quotient = Floor ((ST(0)/ST(1))/2^(ExpDiff-N));
   ST(0) = ST(0) - (ST(1) * Quotient * 2^(ExpDiff-N));
   SW.C2 = 1;
   {SW.C0, SW.C3, SW.C1} = 0;
}
```

Related Instructions

FPREM1, FABS, FRNDINT, FXTRACT, FCHS

rFLAGS Affected

None

x87 Condition Code

x87 Condition Code	Value	Description
C0	M	Set equal to the value of bit 2 of the quotient.
C1	0	There was a stack underflow.
	M	Otherwise, set equal to the value of the least significant bit of the quotient.
C2	1	There is a partial remainder.
	0	There is an incomplete remainder.
C3	M	Set equal to the value of bit 1 of the quotient.
U indicates 'undefined.' M indicates set to 1 or cleared to zero.		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	The source operand was an SNaN value or unsupported format.
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Denormalized-operand exception (DE)	X	X	X	The source operand was a denormal value.
Underflow exception (UE)	X	X	X	The rounded result was too small to fit into the floating-point format of the destination operand.

FPREM1 Floating-Point Partial Remainder

Computes the IEEE Standard 754 remainder obtained by dividing the value in ST(0) by that in ST(1), and stores the result in ST(0). Unlike FPREM, it rounds the integer quotient to the nearest even integer and returns the remainder corresponding to the back multiply of the rounded quotient.

If the exponent difference between ST(0) and ST(1) is less than 64, the instruction computes all integer as well as additional fractional bits of the quotient to do the rounding. The remainder returned is a complete remainder and is less than or equal to one half of the magnitude of the divisor. If the exponent difference is equal to or greater than 64, it computes only the subset of integer quotient bits numbering between 32 and 63, returns the partial remainder, and sets the C2 condition code bit to 1.

Rounding control has no effect. FPREM1 results are exact.

Mnemonic	Opcode	Description
FPREM1	D9 F5	Compute the IEEE standard 754 remainder of the division of ST(0) by ST(1) and store the result in ST(0).

Action

```
ExpDiff = Exponent(ST(0)) - Exponent(ST(1);
if (ExpDiff < 0){
   SW.C2 = 0;
   {SW.C0, SW.C3, SW.C1} = 0;
}
else if (ExpDiff < 64){
   Quotient = Integer obtained by rounding (ST(0)/ST(1))
              to nearest even integer
   ST(0) = ST(0) - (ST(1) * Quotient);
   SW.C2 = 0;
   {SW.C0, SW.C3, SW.C1} = Quotient mod 8;
}
else{
   N = 32 + (ExpDiff mod 32);
   Quotient = Floor ((ST(0)/ST(1))/2^(ExpDiff-N));
   ST(0) = ST(0) - (ST(1) * Quotient * 2^(ExpDiff-N));
   SW.C2 = 1;
   {SW.C0, SW.C3, SW.C1} = 0;
}
```

Related Instructions

FPREM1, FABS, FRNDINT, FXTRACT, FCHS

rFLAGS Affected

None

x87 Condition Code

x87 Condition Code	Value	Description
C0	M	Set equal to the value of bit 2 of the quotient.
C1	0	There was a stack underflow.
	M	Otherwise, set equal to the value of the least significant bit of the quotient.
C2	1	There is a partial remainder.
	0	There is an incomplete remainder.
C3	M	Set equal to the value of bit 1 of the quotient.
U indicates 'undefined.' M indicates set to 1 or cleared to zero.		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
Floating-Point Exception				
Invalid-operation exception (IE)	X	X	X	The source operand was an SNaN value or unsupported format.
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Denormalized-operand exception (DE)	X	X	X	The source operand was a denormal value.
Underflow exception (UE)	X	X	X	The rounded result was too small to fit into the floating-point format of the destination operand.

FPTAN Floating-Point Partial Tangent

Computes the tangent of the radian value in ST(0), stores the result in ST(0), and pushes a value of 1.0 onto the x87 register stack.

The source value must be between -2^{63} and $+2^{63}$ radians. To convert a source value outside of this range to an equivalent acceptable value, use the FPREM instruction to divide the value with a divisor of 2π. If the source value lies outside the specified range, the instruction sets the C2 bit of the x87 status word to 1 and does not change the value in ST(0).

Mnemonic	Opcode	Description
FPTAN	D9 F2	Compute the tangent of ST(0), store the result in ST(0), and push 1.0 onto the x87 register stack.

Related Instructions

FCOS, FPATAN, FSIN, FSINCOS

rFLAGS Affected

None

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	There was an underflow.
	1	There was an overflow.
	0	No round up, if inexact result bit is 1.
	1	Round up, if inexact result bit is 1.
C2	1	ST(0) is out of range, otherwise undefined.
C3	U	
U indicates 'undefined.' M indicates set to 1 or cleared to zero.		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	The source operand was an SNaN value or unsupported format.
Invalid-operation exception (IE) with stack fault (SF)				A stack overflow or underflow occurred.
Denormalized-operand exception (DE)	X	X	X	The source operand was a denormal value.
Underflow exception (UE)	X	X	X	The rounded result was too small to fit into the floating-point format of the destination operand.
Precision exception (PE)	X	X	X	The result could not be represented exactly in the destination format.

FRNDINT Floating-Point Round to Integer

Rounds the value in ST(0) to an integer, depending on the setting of the rounding control (RC) field of the x87 control word, and stores the result in ST(0).

If the initial value in ST(0) is ∞, the instruction does not change ST(0). If the value in ST(0) is not an integer, it sets the precision exception (PE) bit of the x87 status word to 1.

Mnemonic	Opcode	Description
FRNDINT	D9 FC	Round the contents of ST(0) to an integer.

Related Instructions

FABS, FPREM, FXTRACT, FCHS

rFLAGS Affected

None

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	No round up, if inexact result bit (5) is 1.
	1	Round up, if inexact result bit (5) is 1.
C2	U	
C3	U	
U indicates 'undefined.' M indicates set to 1 or cleared to zero.		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	The source operand was an SNaN value or unsupported format.
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Denormalized-operand exception (DE)	X	X	X	The source operand was a denormal value.
Precision exception (PE)	X	X	X	The source operand was not an integral value.

FRSTOR Floating-Point Restore x87 and MMX™ State

Restores the complete x87 state from memory starting at the specified address, as stored by a previous call to FNSAVE. The x87 state occupies 94 or 108 bytes of memory depending on whether the processor is operating in real or protected mode and whether the operand-size attribute is 16-bit or 32-bit. Because the MMX registers are mapped onto the low 64 bits of the x87 floating-point registers, this operation also restores the MMX state.

Unlike the FXRSTOR instruction, FRSTOR checks for pending unmasked floating-point exceptions in the restored image. However, if such a pending unmasked exception is found, the exception does not occur immediately after execution of an FRSTOR instruction. Instead, the processor asserts the FERR# (floating-point error) output signal. The normal rules for x87 exceptions then apply, just as if the exception was created by an x87 instruction. If CRO.NE = 1, the processor vectors to the floating-point exception (#MF) handler at the next non-wait x87 instruction. Otherwise, the processor examines the state of the IGNNE# (ignore numeric error) input signal. If IGNNE# is asserted, the processor executes the next x87 instruction. If IGNNE# is negated, the processor freezes at that instruction boundary until an external interrupt occurs that redirects control to a handler that clears the exception.

For details about the memory image restored by FRSTOR, see “Media and x87 Processor State” in volume 2.

Mnemonic	Opcode	Description
FRSTOR *mem94/108env*	DD /4	Load the x87 state from *mem94/108env*.

Related Instructions

FSAVE, FNSAVE, FXSAVE, FXRSTOR

rFLAGS Affected

None

x87 Condition Code

x87 Condition Code	Value	Description
C0	M	Loaded from memory.
C1	M	Loaded from memory.
C2	M	Loaded from memory.
C3	M	Loaded from memory.

Exceptions (All Modes)

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) was set to 1.
Stack, #SS	X	X	X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking is enabled.
x87 floating-point exception pending, #MF	X	X	X	An x87 floating-point exception was pending.

FSCALE Floating-Point Scale

Multiplies the floating-point value in ST(0) by 2 to the power of the integer portion of the floating-point value in ST(1).

This instruction provides an efficient method of multiplying (or dividing) by integral powers of 2 because, typically, it simply adds the integer value to the exponent of the value in ST(0), leaving the significand unaffected. However, if the value in ST(0) is a denormal value, the mantissa is also modified and the result may end up being a normalized number. Likewise, if overflow or underflow results from a scale operation, the mantissa of the resulting value will be different from that of the source.

The FSCALE instruction performs the reverse operation to that of the FXTRACT instruction.

Mnemonic	Opcode	Description
FSCALE	D9 FD	SCALE ST(0) by ST(1).

Related Instructions

FSQRT, FPREM, FPREM1, FRNDINT, FXTRACT, FABS, FCHS

rFLAGS Affected

None

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	Undefined.
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	No round up, if inexact result bit (5) is 1.
	1	Round up, if inexact result bit (5) is 1.
C2	U	Undefined.
C3	U	Undefined
U indicates 'undefined.' M indicates set to 1 or cleared to zero.		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	The source operand was an SNaN value or unsupported format.
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Denormalized-operand exception (DE)	X	X	X	The source operand was a denormal value.
Overflow exception (OE)	X	X	X	The rounded result was too large to fit into the floating-point format of the destination operand.
Underflow exception (UE)	X	X	X	The rounded result was too small to fit into the floating-point format of the destination operand.
Precision exception (PE)	X	X	X	The result could not be represented exactly in the destination format.

FSIN Floating-Point Sine

Computes the sine of the radian value in ST(0) and stores the result in ST(0).

The source value must be in the range -2^{63} to $+2^{63}$ radians. If the value lies outside this range, the instruction sets the C2 bit in the x87 status word to 1 and does not change the value in ST(0). To convert a source value outside the range -2^{63} and $+2^{63}$ to an equivalent acceptable value, use the FPREM instruction to divide it by 2π.

Mnemonic	Opcode	Description
FSIN	D9 FE	Replace ST(0) with the sine of ST(0).

Related Instructions

FCOS, FPATAN, FSINCOS

rFLAGS Affected

None

x87 Condition Code

x87 Condition Code		Description
C0	U	
C1	0	Numeric underflow is set.
	0	No round up, if inexact result bit is 1.
	1	Round up, if inexact result bit is 1.
C2	1/U	1 if the source value is out of range, otherwise undefined.
C3	U	
U indicates 'undefined.' M indicates set to 1 or cleared to zero.		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) was set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	The source operand was an SNaN value or unsupported format.
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Denormalized-operand exception (DE)	X	X	X	The source operand was a denormal value.
Precision exception (PE)	X	X	X	The result could not be represented exactly in the destination format.

FSINCOS Floating-Point Sine and Cosine

Computes the sine and cosine of the value in ST(0), stores the sine in ST(0), and pushes the cosine onto the x87 register stack. The source value must be in the range -2^{63} to $+2^{63}$ radians.

If the source operand is outside this range, the instruction sets the C2 bit in the x87 status word to 1 and does not change the value in ST(0). To convert a source value outside the range -2^{63} and $+2^{63}$ to an equivalent acceptable value, use the FPREM instruction to divide it by 2π.

Mnemonic	Opcode	Description
FSINCOS	D9 FB	Replace ST(0) with the sine of ST(0) and push the cosine of ST(0) onto the x87 register stack.

Related Instructions

FCOS, FPATAN, FPTAN, FSIN

rFLAGS Affected

None

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Numeric underflow is set.
	0	No round up, if inexact result bit is 1.
	1	Round up, if inexact result bit is 1.
C2	U	
C3	U	
U indicates 'undefined.' M indicates set to 1 or cleared to zero.		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) was set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	The source operand was an SNaN value or unsupported format.
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Denormalized-operand exception (DE)	X	X	X	The source operand was a denormal value.
Underflow exception (UE)	X	X	X	The rounded result was too small to fit into the floating-point format of the destination operand.
Inexact result (5) (precision)	X	X	X	The result could not be represented exactly in the destination format.

FSQRT Floating-Point Square Root

Computes the square root of the value in ST(0) and stores the result in ST(0).

Mnemonic	Opcode	Description
FSQRT	D9 FA	Replace ST(0) with the square root of ST(0).

Related Instructions

FSCALE, FPREM, FPREM1, FRNDINT, FXTRACT, FABS, FCHS

rFLAGS Affected

None

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	No round up, if inexact result bit (5) is 1.
	1	Round up, if inexact result bit (5) is 1.
C2	U	
C3	U	
U indicates 'undefined.' M indicates set to 1 or cleared to zero.		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) was set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	The source operand was an SNaN value or unsupported format.
	X	X	X	Source operand was a negative value (except for -0).
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Denormalized-operand exception (DE)	X	X	X	The source operand was a denormal value.
Precision exception (PE)	X	X	X	The result could not be represented exactly in the destination format.

FST
FSTP
Floating-Point Store Stack Top

Copies the value in ST(0) to the specified floating-point register or memory location.

The FSTP instruction pops the x87 stack after copying the value. The instruction FSTP ST(0) is the same as popping the stack with no data transfer.

If the specified destination is a single-precision or double-precision memory location, the instruction converts the value to the appropriate precision format. It does this by truncating the significand of the source value to the width of the memory location and rounding as specified by the rounding mode determined by the RC field of the x87 control word. It also converts the exponent to the width and bias of the destination format.

If the value is too large for the destination format, the instruction sets the overflow exception (OE) bit of the x87 status word. Then, if the overflow exception is unmasked (OM bit cleared to 0 in the x87 control word), the instruction does not perform the operation.

If the value is a denormal value, the instruction sets the underflow exception (UE) bit in the x87 status word.

If the value is ±0, ±∞, or a NaN, the instruction truncates the least significant bits of the significand and exponent to fit the destination location.

Mnemonic	Opcode	Description
FST *mem32real*	D9 /2	Copy the contents of ST(0) to *mem32real*.
FST *mem64real*	DD /2	Copy the contents of ST(0) to *mem64real.*
FST ST(*i*)	DD D0+*i*	Copy the contents of ST(0) to ST(*i*).
FSTP *mem32real*	D9 /3	Copy the contents of ST(0) to *mem32real* and pop the x87 register stack
FSTP *mem64real*	DD /3	Copy the contents of ST(0) to *mem64real* and pop the x87 register stack.
FSTP *mem80real*	DB /7	Copy the contents of ST(0) to *mem80real* and pop the x87 register stack.
FSTP ST(*i*)	DD D8+*i*	Copy the contents of ST(0) to ST(*i*) and pop the x87 register stack.

Related Instructions

FFREE, FLD, FST, FSTP, FILD, FIST, FISTP, FBLD, FBSTP

rFLAGS Affected

None

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	No round up, if inexact result bit (5) is 1. Otherwise, no other exception flags are set and operand is double-extended-precision format.
	1	Round up, if inexact result bit (5) is 1.
C2	U	
C3	U	
U indicates 'undefined.' M indicates set to 1 or cleared to zero.		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) was set to 1.
Stack, #SS	X	X	X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	Result was located in a nonwritable segment.
			X	A null data segment was used to reference memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking is enabled.
x87 Floating-Point Exception Pending, #MF				

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid-operation exception (IE)	X	X	X	**The source operand was an SNaN value or unsupported format.**
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	**A stack overflow or underflow occurred.**
Overflow exception (OE)	X	X	X	The rounded result was too large to fit into the floating-point format of the destination operand.
Underflow exception (UE)	X	X	X	The rounded result was too small to fit into the floating-point format of the destination operand.
Precision exception (PE)	X	X	X	The result could not be represented exactly in the destination format.

FNSTCW (FSTCW) Floating-Point Store Control Word

Stores an image of the x87 control word in the specified 2-byte memory location. The FNSTCW instruction does not check for possible floating-point exceptions before copying the image of the x87 status register.

Assemblers usually provide an FSTCW macro that expands into the instruction sequence:

```
WAIT                        ; Opcode 9B
FNSTCW destination          ; Opcode D9 /7
```

The WAIT (9Bh) instruction checks for pending x87 exception and calls an exception handler, if necessary. The FNSTCW instruction then stores the state of the x87 control register to the desired destination.

Mnemonic	Opcode	Description
FNSTCW *mem2env*	D9 /7	Copy the x87 control word to *mem2env* without checking for floating-point exceptions.
FSTCW *mem2env*	9B D9 /7	Perform a WAIT (9B) to check for pending floating-point exceptions, then copy the x87 control word to *mem2env*.

Related Instructions

FSTSW, FNSTSW, FNSTCW, FSTENV, FNSTENV

rFLAGS Affected

None

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Numeric underflow is set.
	0	No round up, if inexact result bit is 1.
	1	Round up, if inexact result bit is 1.

x87 Condition Code	Value	Description
C2	U	
C3	U	
U indicates 'undefined.' M indicates set to 1 or cleared to zero.		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) was set to 1.
Stack, #SS	X	X	X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	Result was located in a nonwritable segment.
			X	A null data segment was used to reference memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking is enabled.

FNSTENV (FSTENV) Floating-Point Store x87 Environment

Stores the current x87 environment to memory starting at the specified address, and then masks all floating-point exceptions. The x87 environment consists of the x87 control, status, and tag word registers, the last non-control x87 instruction pointer, the last x87 data pointer, and the opcode of the last completed non-control x87 instruction.

The x87 environment requires a 14-byte or 28-byte area in memory, depending on whether the processor is operating in protected or real mode and whether the operand-size attribute is 16-bit or 32-bit. See "Media and x87 Processor State" in volume 2 for details on how this instruction stores the x87 environment in memory.

The FNSTENV instruction does not check for possible floating-point exceptions before storing the environment. Processor interrupts should be disabled before using this instruction.

Assemblers usually provide an FSTENV macro that expands into the instruction sequence

```
WAIT                    ; Opcode 9B
FNSTENV destination     ; Opcode D9 /6
```

The WAIT (9Bh) instruction checks for pending x87 exceptions and calls an exception handler if necessary. The FNSTENV instruction then stores the state of the x87 environment to the specified destination.

Exception handlers often use these instructions because they provide access to the x87 instruction and data pointers. An exception handler typically saves the environment on the stack. The instructions mask all floating-point exceptions after saving the environment to prevent those exceptions from interrupting the exception handler.

Mnemonic	Opcode	Description
FNSTENV *mem14/28env*	D9 /6	Copy the x87 environment to *mem14/28env* without checking for pending floating-point exceptions, and mask the exceptions.
FSTENV *mem14/28env*	9B D9 /6	Perform a WAIT (9B) to check for pending floating-point exceptions, then copy the x87 environment to *mem14/28env* and mask the floating-point exceptions.

Related Instructions

FLDENV, FSTSW, FNSTSW, FSTCW, FNSTCW, FNSTENV

rFLAGS Affected

None

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Numeric underflow is set.
	0	No round up, if inexact result bit is 1.
	1	Round up, if inexact result bit is 1.
C2	U	
C3	U	
U indicates 'undefined.' M indicates set to 1 or cleared to zero.		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) was set to 1.
Stack, #SS	X	X	X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	Result was located in a nonwritable segment.
			X	A null data segment was used to reference memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking is enabled.

FNSTSW (FSTSW) Floating-Point Store x87 Status Word

Stores the current state of the x87 status word register in either the AX register or a specified two-byte memory location. The image of the status word placed in the AX register always reflects the result after the execution of the previous x87 instruction.

The AX form of the instruction is useful for performing conditional branching operations based on the values of x87 condition flags.

The FNSTSW instruction does not check for possible floating-point exceptions before storing the x87 status word. Processor interrupts should be disabled before using this instruction.

Assemblers usually provide an FSTSW macro that expands into the instruction sequence:

```
WAIT                    ; Opcode 9B
FNSTSW destination      ; Opcode DD /7 or DF E0
```

The WAIT (9Bh) instruction checks for pending x87 exceptions and calls an exception handler if necessary. The FNSTSW instruction then stores the state of the x87 status register to the desired destination.

Mnemonic	Opcode	Description
FNSTSW *mem2env*	DD /7	Copy the x87 status word to *mem12byte* without checking for pending floating-point exceptions.
FNSTSW AX	DF E0	Copy the x87 status word to the AX register without checking for pending floating-point exceptions.
FSTSW *mem2env*	9B DD /7	Perform a WAIT (9B) to check for pending floating-point exceptions, then copy the x87 status word to *mem12byte*.
FSTSW AX	9B DF E0	Perform a WAIT (9B) to check for pending floating-point exceptions, then copy the x87 status word to the AX register.

Related Instructions

FSTCW, FNSTCW, FSTENV, FNSTENV

rFLAGS Affected

None

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Numeric underflow is set.
	0	No round up, if inexact result bit is 1.
	1	Round up, if inexact result bit is 1.
C2	U	
C3	U	
U indicates 'undefined.' M indicates set to 1 or cleared to zero.		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) was set to 1.
Stack, #SS	X	X	X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	Result was located in a nonwritable segment.
			X	A null data segment was used to reference memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking is enabled.

FSUB FSUBP FISUB — Floating-Point Subtract

Subtracts the value in a floating-point register or memory location from the value in a another register and stores the result in that register.

If no operands are specified, the instruction subtracts the value in ST(0) from that in ST(1) and stores the result in ST(1).

If one operand is specified, it subtracts a floating-point or integer value in memory from the contents of ST(0) and stores the result in ST(0).

If two operands are specified, it subtracts the value in ST(0) from the value in another floating-point register or vice versa.

The FSUBP instruction pops the x87 register stack after performing the subtraction.

The no-operand version of the instruction always pops the register stack. In some assemblers, the mnemonic for this instruction is FSUB rather than FSUBP.

The FISUB instruction converts an integer source value to double-extended-precision format before performing the subtraction.

Mnemonic	Opcode	Description
FSUB *mem32real*	D8 /4	Replace ST(0) with ST(0) – *mem32real*.
FSUB *mem64real*	DC /4	Replace ST(0) with ST(0) – *mem64real*.
FSUB ST(0),ST(*i*)	D8 E0+*i*	Replace ST(0) with ST(0) – ST(*i*).
FSUB ST(*i*),ST(0)	DC E8+*i*	Replace ST(*i*) with ST(*i*) – ST(0)
FSUBP	DE E9	Replace ST(1) with ST(1) – ST(0) and pop the x87 register stack.
FSUBP ST(*i*),ST(0)	DE E8+*i*	Replace ST(*i*) with ST(*i*) – ST(0), and pop the x87 register stack.
FISUB *mem16int*	DE /4	Replace ST(0) with ST(0) – *mem16int*.
FISUB *mem32int*	DA /4	Replace ST(0) with ST(0) – *mem32int*.

Related Instructions

FSUBRP, FISUBR, FSUBP, FSUBR, FISUB

rFLAGS Affected

None

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	No round up, if inexact result bit (5) is 1.
	1	Round up, if inexact result bit (5) is 1.
C2	U	
C3	U	
U indicates 'undefined.' M indicates set to 1 or cleared to zero.		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) was set to 1.
Stack, #SS	X	X	X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking is enabled.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	The source operand was an SNaN value or unsupported format.
	X	X	X	The operands were infinities of like sign.
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Denormalized-operand exception (DE)	X	X	X	**The source operand was a denormal value.**
Overflow exception (OE)	X	X	X	The rounded result was too large to fit into the floating-point format of the destination operand.
Underflow exception (UE)	X	X	X	The rounded result was too small to fit into the floating-point format of the destination operand.
Inexact result (5) (precision)	X	X	X	The result could not be represented exactly in the destination format.

FSUBR FSUBRP FISUBR — Floating-Point Subtract Reverse

Subtracts the value in a floating-point register from the value in another register or a memory location, and stores the result in the first specified register. Values in memory can be in single-precision or double-precision floating-point, word integer, or short integer format.

If one operand is specified, the instruction subtracts the value in ST(0) from the value in memory and stores the result in ST(0).

If two operands are specified, it subtracts the value in ST(0) from the value in another floating-point register or vice versa.

The FSUBRP instruction pops the x87 register stack after performing the subtraction.

The no-operand version of the instruction always pops the register stack. In some assemblers, the mnemonic for this instruction is FSUBR rather than FSUBRP.

The FISUBR instruction converts an integer source operand to double-extended-precision format before performing the subtraction.

The FSUBR instructions perform the reverse operations of the FSUB instructions.

Mnemonic	Opcode	Description
FSUBR *mem32real*	D8 /5	Replace ST(0) with *mem32real* - ST(0).
FSUBR *mem64real*	DC /5	Replace ST(0) with *mem64real* - ST(0).
FSUBR ST(0),ST(*i*)	D8 E8+*i*	Replace ST(0) with ST(*i*) - ST(0).
FSUBR ST(*i*),ST(0)	DC E0+*i*	Replace ST(*i*) with ST(0) - ST(*i*)
FSUBRP	DE E1	Replace ST(1) with ST(0) - ST(1) and pop x87 stack.
FISUBR *mem16int*	DE /5	Replace ST(0) with *mem16int* - ST(0).
FISUBR *mem32int*	DA /5	Replace ST(0) with *mem32int* - ST(0).
FSUBRP ST(*i*),ST(0)	DE E0+*i*	Replace ST(*i*) with ST(0) - ST(*i*) and pop x87 stack.

Related Instructions

FSUBRP, FISUBR, FSUB, FSUBP, FISUB

rFLAGS Affected

None

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	No round up, if inexact result bit (5) is 1.
	1	Round up, if inexact result bit (5) is 1.
C2	U	
C3	U	
U indicates 'undefined.' M indicates set to 1 or cleared to zero.		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
Stack, #SS	X	X	X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking is enabled.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	The source operand was an SNaN value or unsupported format.
	X	X	X	The operands were infinities of like sign.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Denormalized-operand exception (DE)	X	X	X	The source operand was a denormal value.
Overflow exception (OE)	X	X	X	The rounded result was too large to fit into the floating-point format of the destination operand.
Underflow exception (UE)	X	X	X	The rounded result was too small to fit into the floating-point format of the destination operand.
Precision exception (PE)	X	X	X	The result could not be represented exactly in the destination format.

FTST Floating-Point Test with Zero

Compares the value in ST(0) with 0.0, and sets the condition code flags in the x87 status word as shown in the x87 Condition Code table below. The instruction ignores the sign distinction between –0.0 and +0.0.

Mnemonic	Opcode	Description
FTST	D9 E4	Compare ST(0) to 0.0.

Related Instructions

FCOM, FCOMPP, FCOMI, FCOMIP, FICOM, FICOMP, FTST, FUCOMI, FUCOMIP, FXAM

rFLAGS Affected

None

x87 Condition Code

C3	C2	C0	Description
0	0	0	ST(0) > 0.0
0	0	1	ST(0) < 0.0
1	0	0	ST(0) = 0.0
1	1	1	Numbers cannot be compared.

x87 Condition Code	Value	Description
C0	0/1	See previous chart for interpretation.
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	If no other flags are set.
C2	0/1	See previous chart for interpretation.
C3	0/1	See previous chart for interpretation.
U indicates 'undefined.' M indicates set to 1 or cleared to zero.		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) was set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	One or both operands were NaN value or unsupported format.
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Denormalized-operand exception (DE)	X	X	X	One or both operands were denormal value.

FUCOM FUCOMP FUCOMPP — Floating-Point Unordered Compare

Compares the value in ST(0) to the value in another x87 register, and sets the condition codes in the x87 status word as shown in the x87 Condition Code table below.

If no source operand is specified, the instruction compares the value in ST(0) to that in ST(1).

After making the comparison, the FUCOMP instruction pops the x87 stack register and the FUCOMPP instruction pops the x87 stack register twice.

The instruction carries out the same comparison operation as the FCOM instructions, but sets the invalid-operation exception (IE) bit in the x87 status word to 1 when either or both operands are an SNaN or are in an unsupported format. If either or both operands is a QNaN, it sets the condition code flags to unordered, but does not set the IE bit. The FCOM instructions, on the other hand, raise an IE exception when either or both of the operands are a NaN value or are in an unsupported format.

Mnemonic	Opcode	Description
FUCOM	DD E1	Compare ST(0) to *S*T(1) and set condition code flags to reflect the results of the comparison.
FUCOM ST(*i*)	DD E0+*i*	Compare ST(0) to ST(*i*) and set condition code flags to reflect the results of the comparison.
FUCOMP	DD E9	Compare ST(0) to *S*T(1), set condition code flags to reflect the results of the comparison, and pop the x87 register stack.
FUCOMP ST(*i*)	DD E8+*i*	Compare ST(0) to ST(*i*), set condition code flags to reflect the results of the comparison, and pop the x87 register stack.
FUCOMPP	DA E9	Compare ST(0) to *S*T(1), set condition code flags to reflect the results of the comparison, and pop the x87 register stack twice.

Related Instructions

FCOM, FCOMPP, FCOMI, FCOMIP, FICOM, FICOMP, FTST, FUCOMI, FUCOMIP, FXAM

rFLAGS Affected

None

x87 Condition Code

C3	C2	C0	Description
0	0	0	ST(0) > source
0	0	1	ST(0) < source
1	0	0	ST(0) = source
1	1	1	numbers cannot be compared

x87 Condition Code	Value	Description
C0	M	Depending on result of the comparison. (See previous table.)
C1	0	0 if numeric underflow is set.
	0	0 indicates no round up, if inexact result bit is 1.
	1	1 indicates round up, if inexact result bit is 1.
C2	M	Depending on result of the comparison. (See previous table.)
C3	M	Depending on result of the comparison. (See previous table.)

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) was set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	One or both operands were SNaN values or have unsupported formats. (Detection of a QNaN value in and of itself does not raise an invalid operand exception.)
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Denormalized-operand exception (DE)	X	X	X	One or both operands were denormal values.

FUCOMI FUCOMIP — Floating-Point Unordered Compare and Set Flags

Compares the contents of ST(0) with the contents of another floating-point register, and sets the zero flag (ZF), parity flag (PF), and carry flag (CF) as shown in the rFLAGS Affected table below.

The FUCOMI and FUCOMIP instructions do not set the invalid-operation exception (IE) bit in the x87 status word for QNaNs.

After completing the comparison, FUCOMIP pops the x87 register stack.

Mnemonic	Opcode	Description
FUCOMI ST(0),ST(*i*)	DB E8+*i*	Compare ST(0) to ST(*i*) and set status flags to reflect the results of the comparison.
FUCOMIP ST(0),ST(*i*)	DF E8+*i*	Compare ST(0) to ST(*i*), set status flags to reflect the results of the comparison, and pop the x87 register stack.

Related Instructions

FCOM, FCOMPP, FCOMI, FCOMIP, FICOM, FICOMP, FTST, FUCOMI, FUCOMIP, FXAM

rFLAGS Affected

Flag	ST(0) > ST(i)	ST(0) < ST(i)	ST(0) = ST(i)	Unordered
ZF	0	0	1	1
PF	0	0	0	1
CF	0	1	0	1

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	If no other flags are set.
C2	U	
C3	U	
U indicates 'undefined.' M indicates set to 1 or cleared to zero.		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
Stack, #SS	X	X	X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking is enabled.

Exception	Real	Virtual 8086	Protected	Cause of Exception
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	One or both operands were NaN values or have unsupported format.
	X	X	X	Register was marked empty.
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Denormalized-operand exception (DE)	X	X	X	One or both operands were denormal values.

FWAIT WAIT — Wait for Unmasked x87 Floating-Point Exceptions

Forces the processor to test for pending unmasked floating-point exceptions before proceeding (FWAIT and WAIT are synonyms). Exception-condition reporting (i.e., assertion of the FERR# output signal) occurs immediately when an error occurs, although the servicing of such an exception occurs when the next x87 non-control instruction is executed (if CR0.NE = 1) or when a later instruction, depending on chip-set delay, is executed (if CR0.NE = 0).

This instruction is useful for insuring that unmasked floating-point exceptions are handled before altering the results of a floating point instruction.

Mnemonic	Opcode	Description
FWAIT	9B	Check for any pending floating-point exceptions.

Related Instructions

WAIT

rFLAGS Affected

None

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	U	
C2	U	
C3	U	
U indicates 'undefined.' M indicates set to 1 or cleared to zero.		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	MP and TS in CR0 were set.

FXAM Floating-Point Examine

Examines the value in ST(0) and sets the C0, C2, and C3 condition code flags in the x87 status word as shown in the x87 Condition Code table below to indicate whether the value is a NaN, infinity, zero, empty, denormal, normal finite, or unsupported value. The instruction also sets the C1 flag to indicate the sign of the value (0 = negative, 1 = positive).

Mnemonic	Opcode	Description
FXAM	D9 E5	Characterize the number in the ST(0) register.

Related Instructions

FCOM, FCOMPP, FCOMI, FCOMIP, FICOM, FICOMP, FTST, FUCOMI, FUCOMIP, FXAM

rFLAGS Affected

None

x87 Condition Code

C3	C2	C1	C0	Meaning
0	0	0	0	+unnormal
0	0	0	1	+NaN
0	0	1	0	-unnormal
0	0	1	1	-NaN
0	1	0	0	+normal
0	1	0	1	+infinite
0	1	1	0	-normal
0	1	1	1	-infinite
1	0	0	0	+0
1	0	0	1	empty

C3	C2	C1	C0	Meaning
1	0	1	0	-0
1	0	1	1	empty
1	1	0	0	+denormal
1	1	1	0	-denormal

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.

FXCH Floating-Point Exchange

Exchanges the value in ST(0) with the value in any other x87 register. If no operand is specified, the instruction exchanges the values in ST(0) and ST(1).

Use this instruction to move a value from an x87 register to ST(0) for subsequent processing by a floating-point instruction that can only operate on ST(0).

Mnemonic	Opcode	Description
FXCH	D9 C9	Exchange the contents of ST(0) and ST(1).
FXCH ST(*i*)	D9 C8+*i*	Exchange the contents of ST(0) and *ST*(*i*).

Related Instructions

FLD, FST, FSTP

rFLAGS Affected

None

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	If no other flags are set.
C2	U	
C3	U	
U indicates 'undefined.' M indicates set to 1 or cleared to zero.		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) was set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.

FXRSTOR Floating-Point Restore XMM, MMX™, and x87 State

Restores the XMM, MMX, and x87 state from memory starting at the specified address. The data loaded from memory is the state information previously saved using the FXSAVE instruction. Restoring data with FXRSTOR that had been previously saved with an FSAVE (rather than FXSAVE) instruction results in an incorrect restoration.

Mnemonic	Opcode	Description
FXRSTOR *mem512env*	0F AE /1	Load the XMM, MMX, and x87 state from *mem512env*.

Unlike the FRSTOR instruction, FXRSTOR does not cause pending unmasked x87 floating-point exceptions in the restored image to be recognized when internal control of x87 exceptions is enabled (CR0.NE = 1). When loading a new environment, use an FWAIT instruction after the FXRSTOR instruction to check for and handle any pending unmasked x87 exceptions.

If the restored MXCSR register contains a set bit in an exception status flag and the corresponding exception mask bit is cleared (indicating an unmasked exception), loading the MXCSR register from memory does not cause a SIMD floating-point exception (#XM).

FXRSTOR executes faster than FRSTOR because it does not restore the x87 pointer registers (last instruction pointer, last data pointer, and last opcode), except in the relatively rare cases in which the exception-summary (ES) bit in the x87 status word is set to 1, indicating that an unmasked x87 exception has occurred.

The architecture supports two memory formats for FXRSTOR, a 512-byte 32-bit legacy format, and a 512-byte 64-bit format. Select the 32-bit or 64-bit format by using the corresponding effective operand size in the FXRSTOR instruction. If software running in 64-bit mode executes an FXRSTOR with a 32-bit operand size (no REX-prefix operand-size override), the 32-bit legacy format is used. If software running in 64-bit mode executes an FXRSTOR with a 64-bit operand size (requires REX-prefix operand-size override), the 64-bit format is used. For details about the memory image restored by FXRSTOR, see "Saving Media and x87 Processor State" in volume 2.

If the operating-system FXSAVE/FXRSTOR support (OSFXSR) bit of CR4 is cleared to 0, the saved image of XMM0–XMM7 and MXCSR is not loaded into the processor. A general-protection exception occurs if there is an attempt to load a non-zero value to the bits in MXCSR that are defined as reserved and cleared (bits 31–16 and bit 6).

Related Instructions

FWAIT, FXSAVE

rFLAGS Affected

None

x87 Condition Code

No condition codes are affected.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The instruction was preceded by LOCK prefix.
				The emulate bit (EM) of CR0 was set to 1.
				The FXSAVE/FSRSTOR instructions are not supported, as indicated by bit 24 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	The memory operand was not aligned on a 16-byte boundary.
			X	A memory address exceeded a data segment limit or was non-canonical.
				An attempt was made to write a non-zero value to reserved bits in MXCSR.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking is enabled.

FXSAVE — Floating-Point Save XMM, MMX™, and x87 State

Stores the XMM, MMX, and x87 state in memory starting at the specified address. A memory location that is not aligned on a 16-byte boundary causes a general-protection exception or, in some cases, an alignment-check exception.

Mnemonic	Opcode	Description
FXSAVE *mem512env*	0F AE /0	Copy the XMM, MMX, and x87 state to *mem512env*.

Unlike FSAVE and FNSAVE, FXSAVE does not alter the x87 tag bits. The contents of the saved MMX/x87 data registers are retained, thus indicating that the registers may be valid (or whatever other value the x87 tag bits indicated before the save). To invalidate the contents of the MMX/x87 data registers after FXSAVE, software must execute an FINIT instruction. Also, FXSAVE (like FNSAVE) does not check for pending unmasked x87 floating-point exceptions. Use an FWAIT instruction for that purpose.

FXSAVE executes faster than FSAVE or FNSAVE because it does not save the x87 pointer registers (last instruction pointer, last data pointer, and last opcode), except in the relatively rare cases in which the exception-summary (ES) bit in the x87 status word is set to 1, indicating that an unmasked x87 exception has occurred.

The architecture supports two memory formats for FXSAVE, a 512-byte 32-bit legacy format, and a 512-byte 64-bit format. Select the 32-bit or 64-bit format by using the corresponding effective operand size in the FXSAVE instruction. If software running in 64-bit mode executes an FXSAVE with a 32-bit operand size (no REX-prefix operand-size override), the 32-bit legacy format is used. If software running in 64-bit mode executes an FXSAVE with a 64-bit operand size (requires REX-prefix operand-size override), the 64-bit format is used. For details about the memory image restored by FXRSTOR, see "Saving Media and x87 Processor State" in volume 2.

If the operating-system FXSAVE/FXRSTOR support (OSFXSR) bit of CR4 is cleared to 0, FXSAVE does not save the image of XMM0–XMM7 and MXCSR. For details about the CR4.OSFXSR bit, see "FXSAVE/FXRSTOR Support (OSFXSR) Bit" in volume 2.

Related Instructions

FINIT, FNSAVE, FRSTOR, FSAVE, FXRSTOR, LDMXCSR, STMXCSR

rFLAGS Affected

None

x87 Condition Code

No condition codes are affected.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	the instruction was preceded by the LOCK prefix. The emulate bit (EM) of CR0 was set to 1. The FXSAVE/FSRSTOR instructions are not supported, as indicated by bit 24 in CPUID standard function 1 or extended function 8000_0001.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X X	The memory operand was not aligned on a 16-byte boundary. A memory address exceeded a data segment limit or was non-canonical. An attempt was made to write a non-zero value to reserved bits in MXCSR.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned-memory data reference was performed while alignment checking is enabled.

FXTRACT Floating-Point Extract Exponent and Significand

Extracts the exponent and significand portions of the floating-point value in ST(0), stores the exponent in ST(1), and pushes the significand onto the x87 register stack. After this operation, the new ST(0) contains a real number with the sign and value of the original significand and an exponent of 3FFFh (biased value for true exponent of zero), and ST(1) contains a real number that is the value of the original value's true (unbiased) exponent.

The FXTRACT instruction is useful for converting a double-extended-precision number to its decimal representation.

If the zero-divide-exception mask (ZM) bit of the x87 control word is set to 1 and the source value is ±0, then the instruction stores ±0 in ST(0) and an exponent value of –∞ in register ST(1).

Mnemonic	Opcode	Description
FXTRACT	D9 F4	Extract the exponent and significand of ST(0), store the exponent in ST(0), and push the significand onto the x87 register stack.

Related Instructions

FABS, FPREM, FRNDINT, FCHS

rFLAGS Affected

None

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	If no other flags are set.
C2	U	
C3	U	
U indicates 'undefined.' M indicates set to 1 or cleared to zero.		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	The source operand was an SNaN value or unsupported format.
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Denormalized-operand exception (DE)	X	X	X	The source operand was a denormal value.
Zero-divide exception (ZE)	X	X	X	ST(0) operand was ±0.

FYL2X — Floating-Point y * Log2 x

Computes (ST(1) $*$ $\log_2$(ST(0))), stores the result in ST(1), and pops the x87 register stack. The value in ST(0) must be greater than 0.

Mnemonic	Opcode	Description
FYL2X	D9 F1	Compute ST(1) * $\log_2$ST(0), store the result in ST(1), and pop the x87 register stack.

If the zero-divide-exception mask (ZM) bit in the x87 control word is set to 1 and ST(0) contains ±0, the instruction returns ∞ with the opposite sign of the value in register ST(1).

Related Instructions

FYL2XP1, F2XM1

rFLAGS Affected

None

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are set.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are set.
	0	No round up, if inexact result bit (5) is 1.
	1	Round up, if inexact result bit (5) is 1.
C2	U	
U indicates 'undefined.' M indicates set to 1 or cleared to zero.		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) was set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	Either operand was a SNaN value or unsupported formats.
	X	X	X	Source operand in register ST(0) was a negative finite value (not -0).
	X	X	X	Source operand in register ST(0) was ±0.
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Denormalized-operand exception (DE)	X	X	X	The source operand was a denormal value.
Zero-divide exception (ZE)	X	X	X	ST(0) operand was ±0.
Overflow exception (OE)	X	X	X	The rounded result was too large to fit into the floating-point format of the destination operand.
Underflow exception (UE)	X	X	X	The rounded result was too small to fit into the floating-point format of the destination operand.
Precision exception (PE)	X	X	X	The result could not be represented exactly in the destination format.

FYL2XP1 Floating-Point y * log2 (X+1)

Computes (ST(1) ∗ $\log_2$(ST(0) + 1.0)), stores the result in ST(1), and pops the x87 register stack. The value in ST(0) must be in the range $-(1 - 2^{½} / 2)$ to $(1 - 2^{½} / 2)$.

Mnemonic	Opcode	Description
FYL2XP1	D9 F9	Compute ST(1) * $\log_2$(ST(0) + 1.0), store the result in ST(1), and pop the x87 register stack.

Related Instructions

FYL2X, F2XM1

rFLAGS Affected

None

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	No round up, if inexact result bit (5) is 1.
	1	Round up, if inexact result bit (5) is 1.
C2	U	
C3	U	
U indicates 'undefined.' M indicates set to 1 or cleared to zero.		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) was set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	Either operand was an SNaN values or unsupported formats.
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Denormalized-operand exception (DE)	X	X	X	An attempt was made to load a denormal single-precision or double-precision floating-point value into an x87 register.
Zero-divide exception (ZE)	X	X	X	The source operand was a denormal value.
Overflow exception (OE)	X	X	X	The rounded result was too large to fit into the floating-point format of the destination operand.
Underflow exception (UE)	X	X	X	The rounded result was too small to fit into the floating-point format of the destination operand.
Precision exception (PE)	X	X	X	The result could not be represented exactly in the destination format.

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	U	
C2	U	
C3	U	
U indicates 'undefined.'		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	MP and TS in CR0 were set.

Index